

RP2350 Datasheet

A microcontroller
by Raspberry Pi

版权说明

© 2023-2025 Raspberry Pi有限公司

本文件依据知识共享署名-禁止演绎 4.0 国际许可协议（CC BY-ND）授权。

部分内容版权归 Synopsys, Inc. © 2019 所有。

版权所有。经许可使用。Synopsys 与 DesignWare 为 Synopsys, Inc. 注册商标。部分内容版权归 Arm Limited 所有，© 2000-2001、2005、2007、2009、2011-2012 及 2016 年。

版权所有。经许可使用。

构建日期：2025-07-2

9 构建版本：d126e9e-clean

法律免责声明

Raspberry Pi 产品的技术及可靠性数据（包括数据表），其文本内容可能不定期更新（以下简称“资源”），由 Raspberry Pi Ltd（以下简称“RPL”）按“现状”提供，明确否认任何明示或默示保证，包括但不限于对适销性和特定用途适用性的默示保证。在适用法律允许的最大范围内，RPL 不对使用该资源所导致的任何直接、间接、附带、特殊、示范性或后果性损害承担责任（包括但不限于采购替代品或服务、使用损失、数据或利润损失，或业务中断），无论基于何种责任理论（合同、严格责任或侵权，包括疏忽与否），即使已被告知可能发生此类损害。

RPL 保留在任何时候对资源或其中描述的产品进行任何增强、改进、修正或其他修改的权利，且无需另行通知。

这些资源专为具备相应设计知识水平的熟练用户而设计。用户对其选择和使用这些资源及其中描述的任何产品应用承担全部责任。用户同意赔偿并使 RPL 免于因其使用资源而产生的所有责任、费用、损害或其他损失。

RPL 授权用户仅可将资源与 Raspberry Pi 产品配合使用。严禁对资源进行任何其他用途。未授予任何 RPL 或第三方知识产权的任何许可。

高风险活动。Raspberry Pi 产品非为在需具备故障安全性能的危险环境中使用而设计、制造或意图使用，例如核设施运营、飞机导航或通信系统、空中交通管制、武器系统或安全关键应用（含生命支持系统及其他医疗设备）。在此类环境中，产品故障可能直接导致死亡、人身伤害或严重的物理或环境损害（“高风险活动”）。RPL 明确否认对高风险活动适用性的任何明示或暗示保证，且不对 Raspberry Pi 产品在高风险活动中的使用或纳入承担任何责任。

树莓派产品依照 RPL 标准条款提供。RPL 对资源的提供不扩展或以其他方式修改 RPL 的标准条款，包括但不限于其中所列的免责声明及保证条款。

目录

版权说明	1
法律免责声明	1
1. 引言	13
1.1. 芯片	14
1.2. 引脚参考	15
1.2.1. 引脚位置	15
1.2.2. 引脚说明	16
1.2.3. GPIO 功能（银行0）	17
1.2.4. GPIO 功能（银行1）	21
1.3. 芯片为何命名为RP2350?	22
1.4. 版本历史	23
2. 系统总线	24
2.1. 总线结构	24
2.1.1. 总线优先级	25
2.1.2. 总线安全过滤	25
2.1.3. 原子寄存器访问	26
2.1.4. APB 桥接	26
2.1.5. 窄带IO寄存器写入	27
2.1.6. 全局独占监控器	28
2.1.7. 总线性能计数器	30
2.2. 地址映射	30
2.2.1. 只读存储器（ROM）	30
2.2.2. 执行代码于闪存（XIP）	30
2.2.3. 静态随机存取存储器（SRAM）	31
2.2.4. APB 寄存器	31
2.2.5. AHB 寄存器	33
2.2.6. 核心本地外设（SIO）	33
2.2.7. Cortex-M33 私有外设	34
3. 处理器子系统	35
3.1. SIO	36
3.1.1. 安全与非安全 SIO	37
3.1.2. CPUID	38
3.1.3. GPIO 控制	39
3.1.4. 硬件自旋锁	41
3.1.5. 处理器间 FIFO（邮箱）	42
3.1.6. 门铃	42
3.1.7. 整数除法器	43
3.1.8. RISC-V 平台定时器	43
3.1.9. TMDS 编码器	44
3.1.10. 插值器	44
3.1.11. 寄存器列表	54
3.2. 中断	82
3.2.1. 不可屏蔽中断（NMI）	83
3.2.2. 中断的进一步阅读	83
3.3. 事件信号（Arm）	84
3.4. 事件信号（RISC-V）	84
3.5. 调试	84
3.5.1. 连接至 SW-DP	85
3.5.2. Arm 调试	86
3.5.3. RISC-V 调试	86
3.5.4. 调试电源域	87
3.5.5. SWD 引脚的软件控制	87
3.5.6. 自托管调试	87
3.5.7. 跟踪	88
3.5.8. 救援重置	90

3.5.9. 安全	91
3.5.10. RP-AP	93
3.6. Cortex-M33 协处理器	100
3.6.1. GPIO 协处理器 (GPIOC)	101
3.6.2. 双精度协处理器 (DCP)	104
3.6.3. 冗余协处理器 (RCP)	112
3.6.4. 浮点运算单元	123
3.7. Cortex-M33 处理器	123
3.7.1. 特性	124
3.7.2. 配置	124
3.7.3. 合规性	128
3.7.4. 程序员模型	131
3.7.5. 寄存器列表	148
3.8. Hazard3 处理器	233
3.8.1. 指令集参考	233
3.8.2. 内存访问	278
3.8.3. 内存保护	279
3.8.4. 中断与异常	282
3.8.5. 调试	285
3.8.6. 自定义扩展	286
3.8.7. 指令周期计数	296
3.8.8. 配置	302
3.8.9. 控制与状态寄存器	304
3.9. Arm/RISC-V 架构切换	335
3.9.1. 自动切换	335
3.9.2. 混合架构组合	336
4. 内存	337
4.1. 只读存储器	337
4.2. 静态随机存取存储器	337
4.2.1. 其他片上存储器	338
4.3. 启动RAM	339
4.3.1. 寄存器列表	339
4.4. 外部闪存和PSRAM (XIP)	340
4.4.1. XIP缓存	341
4.4.2. QSPI存储器接口 (QMI)	345
4.4.3. 流式DMA接口	345
4.4.4. 性能计数器	346
4.4.5. XIP_CTRL 寄存器列表	346
4.4.6. XIP_AUX 寄存器列表	350
4.5. OTP	352
5. 启动只读存储器	353
5.1. 启动只读存储器概念	354
5.1.1. 安全与非安全	354
5.1.2. 分区表	354
5.1.3. 闪存权限	355
5.1.4. 镜像定义	355
5.1.5. 块及块循环	356
5.1.6. 块版本控制	357
5.1.7. A/B 版本	357
5.1.8. 哈希与签名	357
5.1.9. 加载映射	358
5.1.10. 打包二进制文件	358
5.1.11. 防回滚保护	359
5.1.12. 闪存镜像启动	359
5.1.13. 闪存分区启动	360
5.1.14. 镜像内分区表启动	360
5.1.15. 闪存启动槽	360
5.1.16. 闪存更新启动及版本降级	361
5.1.17. 试用功能	362
5.1.18. UF2 目标支持	362

5.1.19. 地址转换	363
5.1.20. 自动架构切换	364
5.2. 处理器控制的启动序列	365
5.2.1. 启动结果	365
5.2.2. 启动序列	366
5.2.3. POWMAN 启动向量	371
5.2.4. 看门狗启动向量	371
5.2.5. RAM 镜像启动	372
5.2.6. OTP 启动	373
5.2.7. 闪存启动	373
5.2.8. BOOTSEL (USB/UART) 启动	374
5.2.9. 启动配置 (OTP)	375
5.3. 在处理器核 1 上启动代码	375
5.4. Bootrom API	376
5.4.1. 定位 API 函数	376
5.4.2. API 函数可用性	378
5.4.3. API 函数返回码	378
5.4.4. API 函数与独占访问权限	379
5.4.5. SDK 对 API 的访问	380
5.4.6. API 函数及只读存储器数据的分类列表	380
5.4.7. API 函数及只读存储器数据的字母顺序列表	382
5.4.8. API 函数清单	383
5.5. USB 大容量存储接口	399
5.5.1. RP2350 驱动	399
5.5.2. UF2 格式详情	400
5.5.3. UF2 目标规则	401
5.6. USB PICOBLOCK 接口	403
5.6.1. 设备识别	403
5.6.2. 接口识别	404
5.6.3. 端点识别	404
5.6.4. PICOBLOCK 命令	405
5.6.5. 控制请求	410
5.7. USB 白标化	412
5.7.1. USB 设备描述符	413
5.7.2. USB 设备字符串	413
5.7.3. USB 配置描述符	413
5.7.4. MSD 驱动	413
5.7.5. UF2 INDEX.HTM 文件	413
5.7.6. UF2 INFO_UF2.TXT 文件	414
5.7.7. SCSI 查询命令	414
5.7.8. 卷标简单示例	414
5.7.9. 卷标深入示例	415
5.8. UART 启动	416
5.8.1. 波特率及时钟要求	416
5.8.2. UART 启动 Shell 协议	416
5.8.3. UART 启动编程流程	417
5.8.4. 恢复卡死的接口	417
5.8.5. UART 启动二进制文件要求	418
5.9. 元数据块详细信息	418
5.9.1. 块及块循环	418
5.9.2. 通用块项	419
5.9.3. 镜像定义项	421
5.9.4. 分区表项	425
5.9.5. 最小可用镜像元数据	428
5.10. 启动示例场景	429
5.10.1. 安全启动	429
5.10.2. 签名映像	430
5.10.3. 打包二进制文件	433
5.10.4. A/B 启动	433
5.10.5. 具所有权分区的 A/B 启动	435

5.10.6. 自定义引导加载程序	437
5.10.7. OTP 引导加载程序	439
5.10.8. 回滚版本及引导加载程序	440
6. 电源	441
6.1. 电源供应	441
6.1.1. 数字 IO 电源 (<code>IOVDD</code>)	441
6.1.2. QSPI IO 电源 (<code>QSPI_IOVDD</code>)	441
6.1.3. 数字核心电源 (<code>DVDD</code>)	441
6.1.4. USB PHY 与 OTP 电源 (<code>USB OTP_VDD</code>)	442
6.1.5. ADC 电源 (<code>ADC_AVDD</code>)	442
6.1.6. 核心电压稳压器输入电源 (<code>VREG_VIN</code>)	442
6.1.7. 芯片内电压稳压器模拟电源 (<code>VREG_AVDD</code>)	442
6.1.8. 电源供应顺序	443
6.2. 电源管理	443
6.2.1. 核心电源域	443
6.2.2. 电源状态	444
6.2.3. 电源状态转换	445
6.3. 核心电压调节器	448
6.3.1. 运行模式	448
6.3.2. 软件控制	449
6.3.3. 电源管理器控制	449
6.3.4. 状态	450
6.3.5. 电流限制	450
6.3.6. 过温保护	450
6.3.7. 应用电路	450
6.3.8. 外部元件及 PCB 布局要求	452
6.3.9. 寄存器列表	457
6.4. 电源管理 (POWMAN) 寄存器	457
6.5. 电源降低策略	488
6.5.1. 顶层时钟门控	489
6.5.2. 睡眠状态	489
6.5.3. 休眠状态	489
6.5.4. 内存外围断电	490
6.5.5. 全内存断电	490
6.5.6. 程序员模型	491
7. 复位	494
7.1. 概述	494
7.2. 与 RP2040 的变更	494
7.3. 芯片级复位	495
7.3.1. 芯片级复位表	495
7.3.2. 芯片级复位目标	496
7.3.3. 芯片级复位源	496
7.4. 系统复位 (上电状态机)	497
7.4.1. 复位序列	498
7.4.2. 寄存器控制	499
7.4.3. 与看门狗的交互	499
7.4.4. 寄存器列表	499
7.5. 子系统复位	503
7.5.1. 概述	503
7.5.2. 程序员模型	503
7.5.3. 寄存器列表	505
7.6. 上电复位与欠压检测	508
7.6.1. 上电复位 (POR)	509
7.6.2. 欠压检测 (BOD)	509
7.6.3. 电源监控器	512
7.6.4. 寄存器列表	512
8. 时钟	513
8.1. 概述	513
8.1.1. RP2350 版本间的变更	514
8.1.2. 时钟源	514

8.1.3. 时钟发生器	518
8.1.4. 频率计数器	522
8.1.5. 复苏 (Resus)	522
8.1.6. 程序员模型	523
8.1.7. 寄存器列表	529
8.2. 晶体振荡器 (XOSC)	554
8.2.1. 概述	554
8.2.2. 相较于 RP2040 的变更	556
8.2.3. 使用方法	556
8.2.4. 启动延迟	556
8.2.5. XOSC 计数器	557
8.2.6. DORMANT 模式	557
8.2.7. 程序员模型	558
8.2.8. 寄存器列表	559
8.3. 环形振荡器 (ROSC)	561
8.3.1. 概述	561
8.3.2. 相较于 RP2040 的变更	562
8.3.3. RP2350 修订版本之间的变更	562
8.3.4. ROSC 与 XOSC 的权衡	562
8.3.5. 频率调整	563
8.3.6. 频率随机化	563
8.3.7. ROSC 分频器	563
8.3.8. 随机数发生器	564
8.3.9. ROSC 计数器	564
8.3.10. DORMANT 模式	564
8.3.11. 寄存器列表	565
8.4. 低功耗振荡器 (LPOSC)	569
8.4.1. 频率精度与校准	569
8.4.2. 使用外部低功率时钟	570
8.4.3. 寄存器列表	570
8.5. 计时发生器	570
8.5.1. 概述	570
8.5.2. 寄存器列表	571
8.6. PLL	575
8.6.1. 概述	575
8.6.2. 相较于 RP2040 的变化	575
8.6.3. PLL 参数计算	576
8.6.4. 配置	580
8.6.5. 寄存器列表	583
9. GPIO.	587
9.1. 概述	587
9.2. 相较于 RP2040 的变化	588
9.3. 复位状态	588
9.4. 功能选择	589
9.5. 中断	594
9.6. 引脚	595
9.6.1. 总线保持器模式	596
9.7. 引脚隔离锁存器	596
9.8. 处理器 GPIO 控制 (SIO)	597
9.9. GPIO 协处理器端口	597
9.10. 软件示例	598
9.10.1. 选择 IO 功能	598
9.10.2. 启用 GPIO 中断	603
9.11. 寄存器列表	604
9.11.1. IO - 用户银行	604
9.11.2. IO - QSPI 银行	760
9.11.3. 引脚控制 - 用户银行	785
9.11.4. 引脚控制 - QSPI 银行	812
10. 安全	816
10.1. 概述 (Arm)	816

10.1.1. 安全启动	816
10.1.2. 加密启动	817
10.1.3. 隔离可信与不可信软件	818
10.2. 处理器安全特性 (Arm)	819
10.2.1. 背景	819
10.2.2. IDAU地址映射.	820
10.3. 概述 (RISC-V)	821
10.4. 处理器安全特性 (RISC-V)	821
10.5. 安全启动启用流程	822
10.6. 访问控制	822
10.6.1. GPIO 访问控制.	823
10.6.2. 总线访问控制	824
10.6.3. 寄存器列表	826
10.7. DMA	867
10.7.1. 通道安全属性	868
10.7.2. 内存保护单元	868
10.7.3. DREQ属性	868
10.7.4. IRQ属性	868
10.8. OTP.	869
10.9. 故障检测器.	869
10.9.1. 工作原理	870
10.9.2. 触发响应	870
10.9.3. 寄存器列表	871
10.10. 工厂测试 JTAG.	874
10.11. 退役	874
11. PIO	876
11.1. 概述	876
11.1.1. 与 RP2040 的变化	877
11.1.2. 程序员模型	878
11.2.1. PIO 程序	879
11.2.2. 控制流	879
11.2.3. 寄存器	881
11.2.4. 自动拉取.	881
11.2.5. 停滞	884
11.2.6. 引脚映射	884
11.2.7. IRQ 标志	884
11.2.8. 状态机之间的交互	885
11.3. PIO汇编器 (pioasm)	885
11.3.1. 指令	885
11.3.2. 值	887
11.3.3. 表达式	887
11.3.4. 注释	888
11.3.5. 标签	888
11.3.6. 指令	888
11.3.7. 伪指令	889
11.4. 指令集	889
11.4.1. 概要	889
11.4.2. JMP	890
11.4.3. WAIT	891
11.4.4. IN	892
11.4.5. OUT	893
11.4.6. PUSH.	894
11.4.7. PULL	895
11.4.8. MOV (至RX)	896
11.4.9. MOV (来自 RX)	897
11.4.10. MOV	898
11.4.11. IRQ.	900
11.4.12. SET	901
11.5. 功能细节	902
11.5.1. Side-set	902

11.5.2. 程序包装	903
11.5.3. FIFO 连接	905
11.5.4. 自动推送与自动拉取	906
11.5.5. 时钟分频器	911
11.5.6. GPIO 映射	911
11.5.7. 强制执行及 EXEC 指令	913
11.6. 示例	915
11.6.1. 双工 SPI	915
11.6.2. WS2812 LED	919
11.6.3. UART 发送	921
11.6.4. UART 接收	923
11.6.5. 曼彻斯特码串行发送与接收	926
11.6.6. 差分曼彻斯特编码 (BMC) 发送与接收	929
11.6.7. I2C	932
11.6.8. PWM	936
11.6.9. 加法运算	938
11.6.10. 进一步示例	939
11.7. 寄存器列表	939
12. 外设	961
12.1. UART	961
12.1.1. 概述	961
12.1.2. 功能描述	962
12.1.3. 操作	964
12.1.4. UART 硬件流控	966
12.1.5. UART DMA 接口	967
12.1.6. 中断	969
12.1.7. 程程序员模型	970
12.1.8. 寄存器列表	972
12.2. I2C	983
12.2.1. 功能特性	984
12.2.2. IP 配置	984
12.2.3. I2C 概述	985
12.2.4. I2C 术语	987
12.2.5. I2C 行为	988
12.2.6. I2C 协议	989
12.2.7. 发送 FIFO 管理及 START、STOP 和 RESTART 生成	993
12.2.8. 多主仲裁	995
12.2.9. 时钟同步	995
12.2.10. 操作模式	996
12.2.11. 突波抑制	1001
12.2.12. 快速模式加速操作	1002
12.2.13. 总线清除功能	1002
12.2.14. IC_CLK 频率配置	1003
12.2.15. DMA 控制器接口	1007
12.2.16. 中断寄存器操作	1008
12.2.17. 寄存器列表	1008
12.3. SPI	1046
12.3.1. 与 RP2040 的变更	1047
12.3.2. 概述	1047
12.3.3. 功能描述	1047
12.3.4. 操作	1050
12.3.5. 寄存器列表	1060
12.4. ADC 与温度传感器	1066
12.4.1. 与 RP2040 的变化	1068
12.4.2. ADC 控制器	1069
12.4.3. SAR ADC	1069
12.4.4. ADC ENOB	1073
12.4.5. INL 与 DNL	1073
12.4.6. 温度传感器	1073
12.4.7. 寄存器列表	1073

12.5. PWM	1076
12.5.1. 概述	1077
12.5.2. 程序员模型	1077
12.5.3. 寄存器列表	1086
12.6. DMA	1094
12.6.1. 与 RP2040 的变化	1095
12.6.2. 配置通道	1096
12.6.3. 触发通道	1098
12.6.4. 数据请求 (DREQ)	1100
12.6.5. 中断	1102
12.6.6. 安全性	1102
12.6.7. 总线错误处理	1105
12.6.8. 附加功能	1107
12.6.9. 示例用例	1108
12.6.10. 寄存器列表	1112
12.7. USB	1141
12.7.1. 概述	1141
12.7.2. RP2040的变更内容	1142
12.7.3. 架构	1144
12.7.4. 程序员模型	1155
12.7.5. 寄存器列表	1159
12.8. 系统定时器	1182
12.8.1. 概述	1182
12.8.2. 计数器	1183
12.8.3. 警报	1183
12.8.4. 程序员模型	1184
12.8.5. 寄存器列表	1188
12.9. 看门狗	1193
12.9.1. 概述	1193
12.9.2. 与RP2040的差异	1193
12.9.3. 看门狗计数器	1193
12.9.4. 控制看门狗复位级别	1194
12.9.5. 暂存寄存器	1194
12.9.6. 程序员模型	1194
12.9.7. 寄存器列表	1196
12.10. 常时开启定时器	1197
12.10.1. 概述	1197
12.10.2. 与RP2040的差异	1198
12.10.3. 访问AON定时器	1198
12.10.4. 使用警报	1198
12.10.5. 选择AON定时器的计时源	1199
12.10.6. 将AON定时器同步至外部1Hz时钟	1201
12.10.7. 使用GPIO的外部时钟或滴答信号	1201
12.10.8. 使用快于1毫秒的滴答信号	1201
12.10.9. 寄存器列表	1202
12.11. HSTX	1202
12.11.1. 数据FIFO	1203
12.11.2. 输出移位寄存器	1203
12.11.3. 位交叉开关	1204
12.11.4. 时钟发生器	1205
12.11.5. 命令扩展器	1206
12.11.6. PIO与HSTX耦合模式	1208
12.11.7. 控制寄存器列表	1208
12.11.8. FIFO寄存器列表	1212
12.12. TRNG	1212
12.12.1. 概述	1212
12.12.2. 配置	1213
12.12.3. 操作	1213
12.12.4. 注意事项	1214
12.12.5. 寄存器列表	1215

12.13. SHA-256 加速器	1221
12.13.1. 消息填充	1222
12.13.2. 吞吐量	1222
12.13.3. 数据大小与字节序	1222
12.13.4. DMA DREQ 接口	1222
12.13.5. 寄存器列表	1223
12.14. QSPI 存储接口 (QMI)	1226
12.14.1. 概述	1226
12.14.2. QSPI 传输	1228
12.14.3. 时序	1231
12.14.4. 地址转换	1234
12.14.5. 直接模式	1235
12.14.6. 寄存器列表	1236
12.15. 系统控制寄存器	1249
12.15.1. SYSINFO	1249
12.15.2. SYSCFG	1251
12.15.3. TBMAN	1254
12.15.4. BUSCTRL	1255
13. OTP	1268
13.1. OTP 地址映射	1268
13.1.1. 保护读取	1269
13.2. 背景：OTP IP 细节	1269
13.3. 背景：OTP 硬件架构	1270
13.3.1. 锁存衬垫	1270
13.3.2. 外部接口	1271
13.3.3. OTP 启动振荡器	1272
13.3.4. 上电状态机	1272
13.4. 关键标志	1273
13.5. 页锁	1274
13.5.1. 锁定进度	1274
13.5.2. OTP 访问密钥	1275
13.5.3. OTP 中的锁定编码	1276
13.5.4. 特殊页面	1276
13.5.5. 空白设备权限	1276
13.6. 误差纠正码 (ECC)	1277
13.6.1. 极性位修复 (BRP)	1277
13.6.2. 改进型汉明ECC	1278
13.7. 设备退役 (RMA)	1279
13.8. 镜像漏洞	1279
13.8.1. 最佳实践	1279
13.8.2. 干扰数据	1280
13.9. 寄存器列表	1280
13.10. 预定义OTP数据位置	1292
14. 电气及机械特性	1327
14.1. QFN-60封装	1327
14.1.1. 热特性	1328
14.1.2. 推荐PCB封装尺寸	1328
14.2. QFN-80封装	1328
14.2.1. 热特性	1329
14.2.2. 推荐PCB封装尺寸	1329
14.3. 封装内闪存	1330
14.4. 封装标记	1331
14.5. 存储条件	1331
14.6. 焊接曲线	1331
14.7. 合规性	1333
14.8. 引脚排列	1333
14.8.1. 引脚位置	1333
14.8.2. 引脚定义	1335
14.9. 电气规格	1338
14.9.1. 绝对最大额定值	1338

14.9.2. 静电放电性能	1339
14.9.3. 热性能	1339
14.9.4. IO电气特性	1339
14.9.5. 电源供应	1343
14.9.6. 核心电压调节器	1344
14.9.7. 功耗	1345
附录A：寄存器字段类型	1349
RP2040的变更	1349
标准类型	1349
读写：	1349
只读：	1349
只写：	1349
清除类型	1349
SC:	1349
WC:	1349
FIFO类型	1350
RWF:	1350
RF:	1350
WF:	1350
附录B：本文件中使用的单位	1351
内存与存储容量	1351
传输速率	1351
物理量	1351
单位前缀	1353
数字分隔符	1353
附录C：硬件修订历史	1354
RP2350 A2	1354
RP2350 A3	1354
硬件变更	1354
Bootrom 变更	1355
RP2350 A4	1355
硬件变更	1355
Bootrom 变更	1356
附录E：勘误表	1357
访问控制	1357
RP2350-E3	1357
Bootrom	1357
RP2350-E10	1357
RP2350-E13	1358
RP2350-E14	1358
RP2350-E15	1359
RP2350-E18	1359
RP2350-E19	1360
RP2350-E20	1360
RP2350-E21	1361
RP2350-E22	1362
RP2350-E23	1362
RP2350-E24	1362
RP2350-E25	1363
总线结构	1363
RP2350-E27	1363
DMA	1364
RP2350-E5	1364
RP2350-E8	1365
GPIO	1365
RP2350-E9	1366
Hazard3	1368
RP2350-E4	1368
RP2350-E6	1369
RP2350-E7	1369

OTP	1370
RP2350-E16	1370
RP2350-E17	1371
RP2350-E28	1371
RCP	1372
RP2350-E26	1372
SIO	1373
RP2350-E1	1373
RP2350-E2	1373
XIP	1374
RP2350-E11	1374
USB	1375
RP2350-E12	1375
附录 H: 文档发布历史	1377
2025年 7月29日	1377
2025年2月20日	1377
2024年 12月4日	1377
2024年10月16日	1377
2024年10月15日	1377
2024年9月6日	1377
2024年8月8日	1378

第1章 引言

RP2350 是 Raspberry Pi 推出的新一代微控制器系列，较 RP2040 实现了重大提升。主要特性包括：

- 双核 Cortex-M33 或 Hazard3 处理器，主频150 MHz
- 520 kB片上SRAM，分为10个独立存储区
- 8 kB一次性可编程存储（OTP）
- 通过专用QSPI总线支持最多16 MB外部QSPI闪存或PSRAM
- 通过可选第二芯片选择支持额外16 MB闪存或PSRAM
- 片上开关模式电源，生成核心电压
- 睡眠状态下可选低静态电流LDO模式
- 2个片上锁相环（PLL），用于内部或外部时钟生成
- GPIO支持5 V耐受（加电）及3.3 V故障安全（断电）
- 安全特性：
 - 可选启动签名，由片上掩膜ROM强制执行，密钥指纹存于OTP
 - 用于可选启动解密密钥的保护性OTP存储
 - 基于Arm或RISC-V安全/特权级的全局总线过滤
 - 外设、GPIO与DMA通道可单独分配至安全域
 - 针对故障注入攻击的硬件防护机制
 - 硬件SHA-256加速器
- 外设：
 - 2 × UARTs
 - 2 × SPI 控制器
 - 2 × I2C 控制器
 - 24×PWM 通道
 - USB 1.1 控制器及 PHY，支持主机和设备功能
 - 12×PIO 状态机
 - 1 × HSTX 外设

表1显示了RP2350系列设备，包括带有或不带有封装内闪存的QFN-80（10 × 10 毫米）和QFN-60（7 × 7 毫米）封装选项。

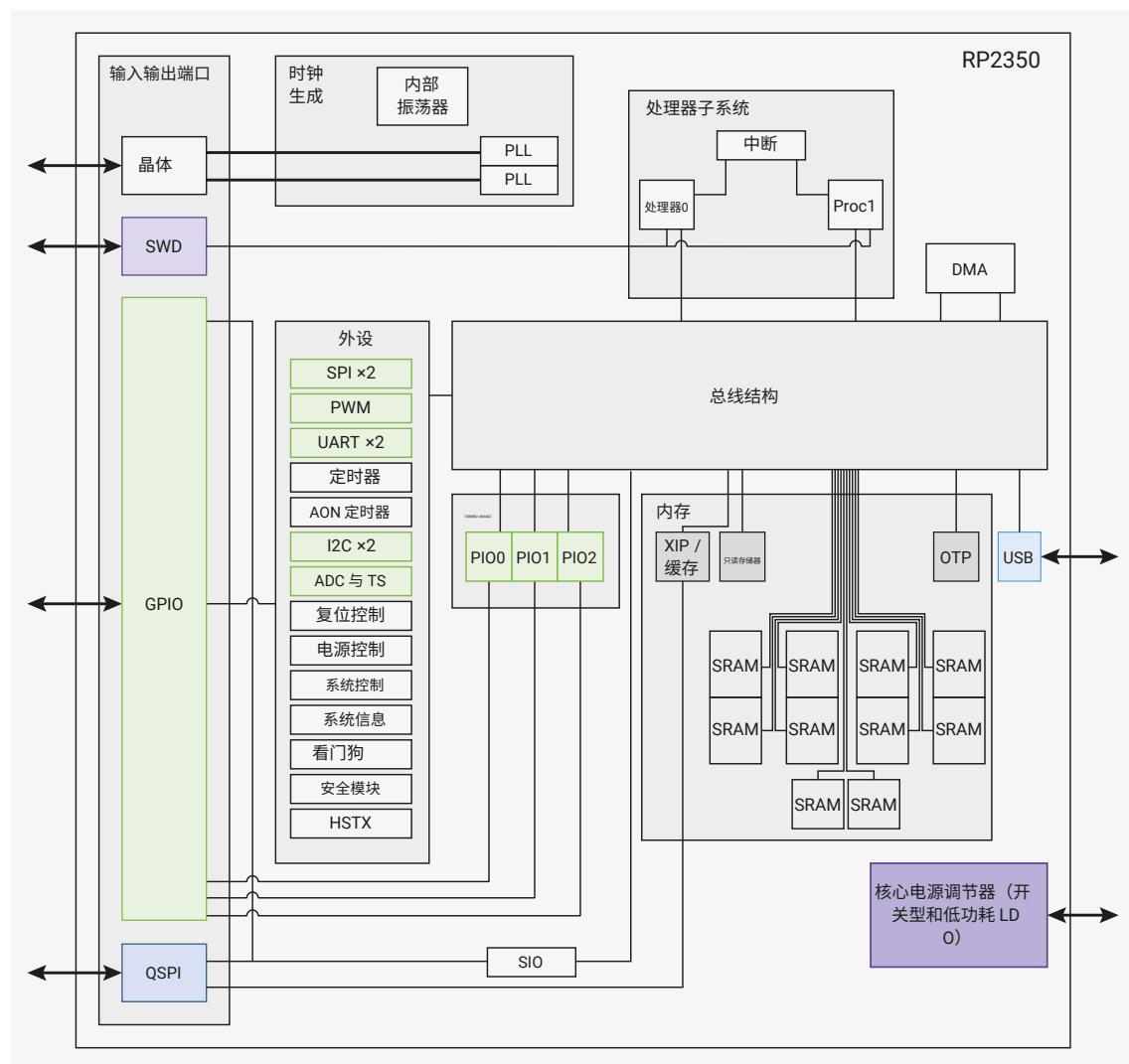
表1. RP2350
设备系列

产品	封装	内部闪存	GPIO	模拟输入
RP2350A	QFN-60	无	30	4
RP2350B	QFN-80	无	48	8
RP2354A	QFN-60	2 MB	30	4
RP2354B	QFN-80	2 MB	48	8

1.1. 芯片

双核 Cortex-M33 或 Hazard3 处理器通过 AHB 和 APB 总线结构访问 RP2350 的内存及外设。

图 1. RP2350 芯片系统概述



代码可通过执行就地子系统（XIP）中的专用QSPI存储接口直接从外部存储器执行。缓存显著提升了XIP的性能。闪存和RAM均可通过该接口连接。

调试功能通过SWD接口提供。此功能允许外部主机加载、运行、暂停及检查系统运行的软件，或配置执行跟踪输出。

内部SRAM可包含代码或数据。地址空间为单一的520 kB区域，物理上分为10个银行，以支持来自不同管理器的并行访问。所有SRAM均支持单周期访问。

高带宽系统DMA减轻了处理器的重复数据传输负担。

GPIO引脚可通过单周期IO（SIO）直接驱动，或由多种专用逻辑功能如硬件SPI、I2C、UART及PWM驱动。可编程IO控制器（PIO）提供更多样的IO功能，或扩展固定功能外设的数量。

带有嵌入式PHY的USB控制器在软件控制下提供FS/LS主机或设备连接功能。

四个或八个ADC输入（视封装尺寸而定）与GPIO引脚共享。

两个PLL为USB或ADC提供固定的48 MHz时钟，并为系统提供最高150 MHz的灵活时钟。晶体振荡器为PLL提供精确的参考时钟。

内部电压调节器为核心电压供电，因此通常只需提供IO电压。它作为一个

开关模式降压转换器在系统唤醒时工作，提供最高200 mA的可变输出电压；系统睡眠时可切换至低静态电流的LDO模式，提供最高1 mA的状态保持电流。

系统设有低功耗状态，未使用的逻辑电路会被断电，支持通过定时器或IO事件唤醒。
掉电期间SRAM保持的容量可配置。

内部8 kB一次性可编程存储器（OTP）存储芯片信息，如唯一标识符，可用于配置硬件及BootROM安全功能，也可编程用户提供的代码和数据。

内置启动只读存储器实现了从闪存或OTP的直接启动，以及通过USB或UART的串行启动。所有启动介质均支持代码签名强制执行，使用注册于内部OTP存储的密钥指纹。OTP还可存储加密启动的解密密钥，防止闪存内容被外部读取。

RISC-V架构支持通过动态替换Cortex-M33（Armv8-M）处理器为Hazard3（RV32IMAC+）处理器实现。所有RP2350系列设备均支持这两种架构。RISC-V内核支持通过SWD调试，且可使用与Arm内核相同的SDK进行编程。

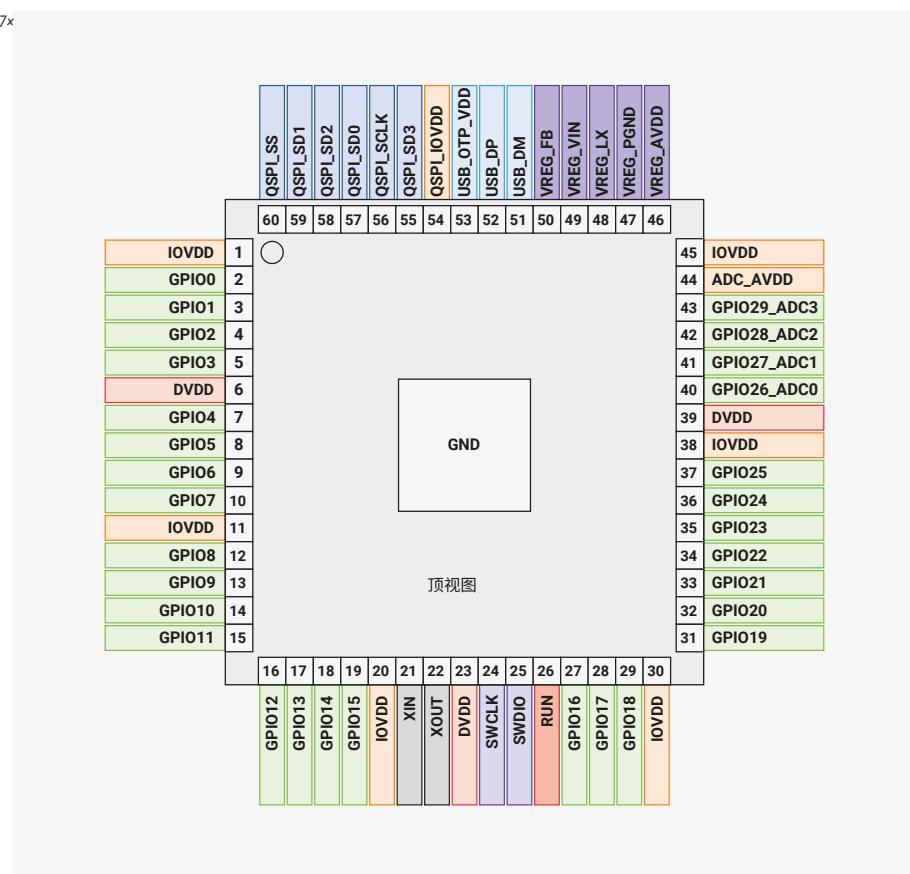
1.2. 引脚参考

本节提供引脚布局及引脚功能的快速参考。详尽信息，包括电气规格与封装图纸，详见第14章。

1.2.1. 引脚位置

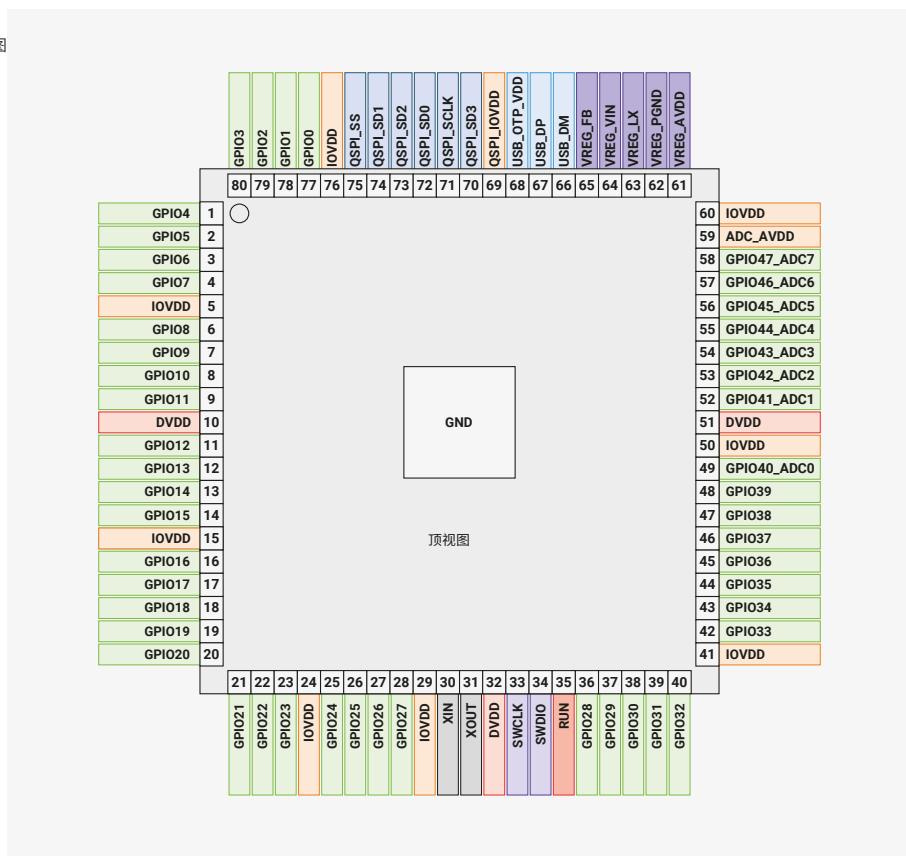
1.2.1.1. QFN-60 (RP2350A)

图2。RP2350 QFN-60 7x
7毫米引脚分布
(缩小的ePad尺寸
)



1.2.1.2. QFN-80 (RP2350B)

图3. RP2350
QFN-80封装引脚分布图
10x10毫米（缩小的
ePad尺寸）



1.2.2. 引脚说明

表2。各引脚功能
简述如下。完整
电气规格详见第1
4章。

名称	描述
GPIOx	通用数字输入和输出。RP2350能够将多个内部外设之一连接至每个GPIO，或通过软件直接控制GPIO。
GPIOx/ADCy	通用数字输入和输出，具备模拟到数字转换功能。RP2350的ADC内置模拟多路复用器，可选择上述引脚中的任意一个采样电压。
QSPIx	用于连接SPI、双SPI或四SPI闪存或PSRAM设备的接口，支持原地执行（execute-in-place）。若不用于闪存访问，这些引脚亦可作为软件控制的GPIO使用。
USB_DM 和 USB_DP	USB控制器，支持全速设备及全速/低速主机模式。每个引脚需串联一个27Ω终端电阻，但总线上的上拉和下拉由内部提供。若不使用USB，这些引脚可作为软件控制的GPIO。
XIN 和 XOUT	将晶体连接至RP2350的晶体振荡器引脚。XIN 亦可用作单端 CMOS 时钟输入，XOUT 应断开连接。USB 引导加载程序默认使用 12MHz 晶振或 12MHz 时钟输入，但可通过 OTP 进行配置。
RUN	全局异步复位引脚。置低复位，置高运行。如无外部复位需求，该引脚可直接连接至 IOVDD。
SWCLK 和 SWDIO	访问内部串行线调试多点总线。提供对双处理器的调试访问，并支持代码下载。
GND	单个外部地线连接，焊接至 RP2350 芯片内部多个接地焊盘。

名称	描述
IOVDD	数字 GPIO 电源，标称电压范围为 1.8V 至 3.3V。
USB OTP VDD	内部 USB 全速 PHY 及 OTP 存储电源，标称电压为 3.3V。
ADC_AVDD	模数转换器电源，标称电压为 3.3V。
QSPI_IOVDD	QSPI IO 电源，标称电压范围为 1.8V 至 3.3V。
VREG_AVDD	内部核心电压调节器的模拟电源，标称电压为 3.3V
VREG_PGND	内部核心电压调节器的电源地连接，需外部接地
VREG_LX	内部核心电压调节器的开关模式输出，连接至外部电感最大电流 200 mA，滤波后标称电压为 1.1V
VREG_VIN	内部核心电压调节器的电源输入，标称电压范围为 2.7V 至 5.5V
VREG_FB	内部核心电压调节器的电压反馈，连接至滤波后的 VREG 输出（例如，若调节器用于供电 DVDD，则连接至 DVDD）
DVDD	数字核心电源，标称电压为 1.1V。必须外部连接，连接方式可为电压调节器输出或外部板级电源

1.2.3. GPIO功能（银行0）

每个独立GPIO引脚均可通过以下定义的GPIO功能连接至内部外设部分内部外设连接分布于多个位置，以提供系统级灵活性SIO、PIO0、PIO1和PIO2可连接所有GPIO引脚，由软件（或软件控制的状态机）控制，能实现多种功能

表3。通用输入/输出 (GPIO) 第0组功能

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
0		SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB 过流检测	
1		SPI0 片选 (CSn)	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM0 B	SIO	PIO0	PIO1	PIO2	TRACECLK	USB VBUS 检测	
2		SPI0 时钟 (SCK)	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM1 A	SIO	PIO0	PIO1	PIO2	TRACEDATA0	USB VBUS 使能	UART0 TX
3		SPI0 发送 (TX)	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM1 B	SIO	PIO0	PIO1	PIO2	TRACEDATA1	USB 过流检测	UART0 接收 (RX)
4		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	TRACEDATA2	USB VBUS 检测	
5		SPI0 片选 (CSn)	UART1 RX	I2C0 时钟线 (SCL)	PWM2 B	SIO	PIO0	PIO1	PIO2	TRACEDATA3	USB VBUS 使能	
6		SPI0 时钟 (SCK)	UART1 CTS	I2C1 数据线 (SDA)	PWM3 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 TX
7		SPI0 发送 (TX)	UART1 RTS	I2C1 时钟线 (SCL)	PWM3 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART1 RX
8		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS 使能	
9		SPI1 CSn	UART1 RX	I2C0 时钟线 (SCL)	PWM4 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	
10		SPI1 SCK	UART1 CTS	I2C1 数据线 (SDA)	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART1 TX
11		SPI1 TX	UART1 RTS	I2C1 时钟线 (SCL)	PWM5 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART1 RX
12	HSTX	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB 过流检测	
13	HSTX	SPI1 CSn	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM6 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB VBUS 检测	
14	HSTX	SPI1 SCK	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM7 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS 使能	UART0 TX
15	HSTX	SPI1 TX	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM7 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB 过流检测	UART0 接收 (RX)
16	HSTX	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
17	HSTX	SPI0 片选 (CSn)	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM0 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
18	HSTX	SPI0 时钟 (SCK)	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM1 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART0 TX
19	HSTX	SPI0 发送 (TX)	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM1 B	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS 检测	UART0 接收 (RX)
20		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB VBUS 使能	
21		SPI0 片选 (CSn)	UART1 RX	I2C0 时钟线 (SCL)	PWM2 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB 过流检测	
22		SPI0 时钟 (SCK)	UART1 CTS	I2C1 数据线 (SDA)	PWM3 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS 检测	UART1 TX

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
23		SPI0 发送 (TX)	UART1 RTS	I2C1 时钟线 (SCL)	PWM3 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB VBUS 使能	UART1 RX
24		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT2	USB 过流检测	
25		SPI1 CSn	UART1 RX	I2C0 时钟线 (SCL)	PWM4 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT3	USB VBUS 检测	
26		SPI1 SCK	UART1 CTS	I2C1 数据线 (SDA)	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART1 TX
27		SPI1 TX	UART1 RTS	I2C1 时钟线 (SCL)	PWM5 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 RX
28		SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
29		SPI1 CSn	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM6 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
GPIO 30 至 47 仅限 QFN-80 型:												
30		SPI1 SCK	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM7 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART0 TX
31		SPI1 TX	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM7 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART0 接收 (RX)
32		SPI0 RX	UART0 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
33		SPI0 片选 (CSn)	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM8 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	
34		SPI0 时钟 (SCK)	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM9 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART0 TX
35		SPI0 发送 (TX)	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM9 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART0 接收 (RX)
36		SPI0 RX	UART1 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	
37		SPI0 片选 (CSn)	UART1 RX	I2C0 时钟线 (SCL)	PWM10 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
38		SPI0 时钟 (SCK)	UART1 CTS	I2C1 数据线 (SDA)	PWM11 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART1 TX
39		SPI0 发送 (TX)	UART1 RTS	I2C1 时钟线 (SCL)	PWM11 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 RX
40		SPI1 RX	UART1 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
41		SPI1 CSn	UART1 RX	I2C0 时钟线 (SCL)	PWM8 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
42		SPI1 SCK	UART1 CTS	I2C1 数据线 (SDA)	PWM9 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 TX
43		SPI1 TX	UART1 RTS	I2C1 时钟线 (SCL)	PWM9 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART1 RX
44		SPI1 RX	UART0 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
45		SPI1 CSn	UART0 接收 (RX)	I ₂ C0 时钟线 (SCL)	PWM10 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	
46		SPI1 SCK	UART0 清除发送 (CTS)	I ₂ C1 数据线 (SDA)	PWM11 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART0 TX
47		SPI1 TX	UART0 请求发送 (RTS)	I ₂ C1 时钟线 (SCL)	PWM11 B	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS 使能	UART0 接收 (RX)

表 4. GPIO 组 0
功能说明

功能名称	描述
SPIx	将内部 PL022 SPI 外设之一连接至 GPIO
UARTx	将内部 PL011 UART 外设之一连接至 GPIO
I2Cx	将内部 DW I2C 外设之一连接至 GPIO
PWMx A/B	将 PWM 片段连接至 GPIO。共有十二个 PWM 片段，每个包含两个输出通道（A/B）。B 引脚亦可用作输入，用于频率及占空比测量。
SIO	通过单周期 IO (SIO) 模块进行 GPIO 的软件控制。处理器驱动 GPIO 时必须选择 SIO 功能 (F5)，但输入端始终连接，因而软件可随时检测 GPIO 状态。
PIOx	将可编程IO模块 (PIO) 之一连接到GPIO。PIO能够实现多种接口，并配备自身的内部引脚映射硬件，允许在bank 0 GPIO上灵活配置数字接口。要使PIO驱动GPIO，必须选择PIO功能 (F6、F7、F8)，但输入端始终连接，因此PIO可持续监测所有引脚的状态。
HSTX	将高速传输外设 (HSTX) 连接到GPIO。
时钟 GPINx	通用时钟输入。可路由至RP2350的多个内部时钟域，例如为AON定时器提供1Hz时钟，或连接至内部频率计数器。
时钟 GPOUTx	通用时钟输出。可将多个内部时钟（包括PLL输出）驱动至GPIO，支持可选的整数分频。
TRACECLK, TRACEDATAx	CoreSight TPIU 执行追踪输出，来自 Cortex-M33 处理器（仅限 Arm）
USB 过流检测 / VBUS 电源线 检测 / VBUS 电源线使能	内部 USB 控制器的 USB 电源控制信号
QMI CS1n	QSPI 总线辅助芯片选择信号，支持从额外的闪存或 PSRAM 设备执行就地运行

i 注意

GPIO 0 至 29 在所有封装型号中均可用。GPIO 30 至 47 仅在 QFN-80 (RP2350B) 封装中可用。

i 注意

模拟输入在 QFN-60 封装 (RP2350A) 的 GPIO 26 至 29 上可用，共计四个输入；在 QFN-80 封装 (RP2350B) 的 GPIO 40 至 47 上可用，共计八个输入。

1.2.4. GPIO功能（银行1）

GPIO 功能还适用于一般用于闪存就地执行的六个专用 QSPI 引脚及 USB DP/DM 引脚。根据具体使用情况，这些接口可能可用作通用用途，例如，如果 RP2350 从内部只读存储器启动，或通过 SWD 进行外部控制，则 QSPI 引脚可能不需要用于代码执行。

表 5. GPIO 库 1
功能

引脚	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
USB DP			UART1 TX	I2C0 SDA		SIO						

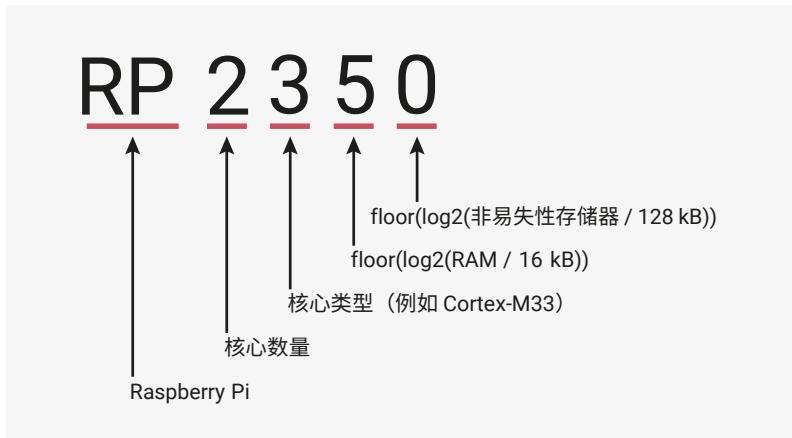
引脚	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
USB DM			UART1 RX	I2C0 时钟线 (SCL)		SIO						
QSPI SCK	QMI SCK		UART1 CTS	I2C1 数据线 (SDA)		SIO						UART1 TX
QSPI CSn	QMI CS0n		UART1 RTS	I2C0 时钟线 (SCL)		SIO						UART1 RX
QSPI SD0	QMI SD0		UART0 TX	I2C0 SDA		SIO						
QSPI SD1	QMI SD1		UART0 接收 (RX)	I2C0 时钟线 (SCL)		SIO						
QSPI SD2	QMI SD2		UART0 清除发送 (CTS)	I2C1 数据线 (SDA)		SIO						UART0 TX
QSPI SD3	QMI SD3		UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)		SIO						UART0 接收 (RX)

表6. GPIO 组 1
功能描述

功能名称	描述
UARTx	将内部 PL011 UART 外设之一连接至 GPIO
I2Cx	将内部 DW I2C 外设之一连接至 GPIO
SIO	通过单周期 IO (SIO) 模块进行 GPIO 的软件控制。处理器驱动 GPIO 时必须选择 SIO 功能 (F5)，但输入端始终连接，因而软件可随时检测 GPIO 状态。
QMI	QSPI 存储器接口外设，用于从外部 QSPI 闪存或 PSRAM 存储器设备执行就地操作。

1.3. 芯片为何命名为RP2350?

图 4。RP2350 芯片名称说明。



RP2350 后缀数字含义如下：

1. 处理器核心数量
 - 2 表示双核系统 2。大致处理器

类型

- 3 表示 Cortex-M33 或 Hazard3

3. 内部存储容量： $\log_2 \frac{RAM}{16kB}$

- 5 表示至少 $2^5 \times 16 kB = 512 kB$

◦ RP2350 具有 520 kB 主系统 SRAM

4. 内部存储容量： $\log_2 \frac{nonvolatile}{128kB}$ (若无板载非易失性存储，则为 0)

- RP2350 采用外部闪存
- RP2354 具备 $2^4 \times 128 \text{ kB} = 2 \text{ MB}$ 内部闪存

1.4. 版本历史

表 7 列出了 RP2350 版本。后续版本修正了早期版本中的缺陷。有关版本间变更的详细信息，请参见附录 C。亦请参阅产品变更通知 (PCN) 28。

表 7. RP2350
版本历史

版本	用途
A0	内部开发
A1	内部开发
A2	初始发布
A3	内部开发、样品及有限量生产
A4	生产版本

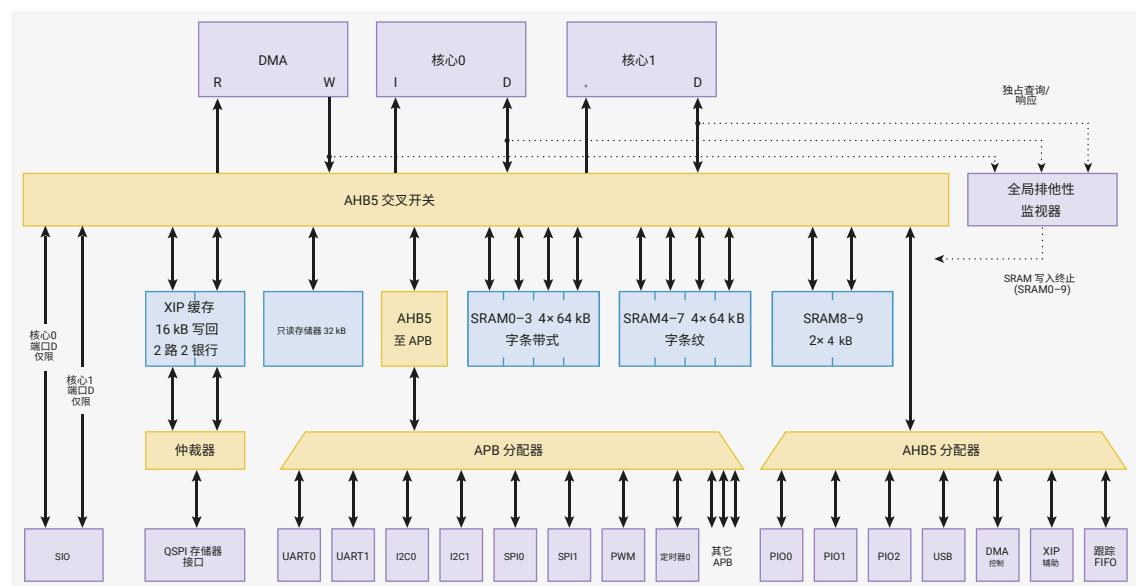
第2章 系统总线

2.1. 总线结构

RP2350总线结构在芯片内进行地址和数据的路由。

图5展示了总线结构的高层次架构。主AHB5交叉开关在其6个上游端口与17个下游端口之间路由地址和数据，每个周期最多可执行六次总线传输。所有数据通路宽度均为32位。存储器连接至主交叉开关的多个专用端口，以实现最佳存储带宽。高带宽AHB外设共享交叉开关上的一个端口。APB桥接器提供对系统控制寄存器及低带宽外设的访问。SIO外设通过每个处理器的专用路径进行访问。

图5. RP2350总线
结构概览。



总线结构连接6个AHB5管理器，即生成地址的总线端口：

- 核心0：指令端口（取指）和数据端口（加载/存储访问）
- 核心1：指令端口（取指）和数据端口（加载/存储访问）
- DMA控制器：读端口，写端口

以下13个下游端口可被所有6个上游端口对称访问：

- 启动只读存储器（1个端口）
- XIP（2个端口，条纹化）
- SRAM（10个端口，分条）

此外，以下两个端口仅供处理器负载/存储及DMA读写使用：

- 1个用于快速AHB5外设的共享端口：PIO0、PIO1、PIO2、USB、DMA控制寄存器、XIP DMA FIFO、HSTX FIFO，CoreSight追踪DMA FIFO
- 1个用于APB桥接的端口，连接所有APB外设及控制寄存器

● 注意

外设指令取指物理断开，以避免此 IDAU-Exempt 区域同时成为非安全可写及安全可执行区域。包括 USB RAM、OTP 和启动 RAM。详见第 10.2.2 节。

SIO 模块，原连接至 RP2040 上的 Cortex-M0+ IOPORT，提供两个 AHB 端口，每个端口专门支持单核的负载/存储访问。

六个管理器可同时访问任意六个不同的交叉开关端口。因此，在 150 MHz 系统时钟下，最大持续总线带宽为 3.6 GB/s。

2.1.1. 总线优先级

主AHB5交叉开关实现了两级总线优先级方案。优先级等级分别针对核心0、核心1、DMA读和DMA写进行配置，使用BUS CTRL寄存器块中的BUS_PRIORITY寄存器。

当下游从属设备同时接收多个访问请求时，端口会优先服务高优先级（优先级等级1）管理器，再处理低优先级（优先级等级0）管理器的请求。如果所有请求均来自相同优先级等级的管理器，端口将采用轮询方式依次授予各管理器访问权限。

● 注意

优先级仲裁仅适用于多个管理器在同一周期试图访问同一从属设备的情况。

当多个管理器访问不同的从属设备时，例如不同的SRAM银行，请求将同时被处理。

具有零等待状态的从属设备可以在每个系统时钟周期内被访问一次。访问具有零等待状态的从属设备（例如SRAM）时，高优先级管理器不会因低优先级管理器的访问而产生延迟。该设计保证实时应用中的延迟和吞吐量。然而，这也意味着低优先级管理器可能会阻塞，直到有空闲周期为止。

2.1.2. 总线安全过滤

所有连接至下游AHB或APB外设的总线交汇点均设有总线安全过滤器，该过滤器依据ACCESSCTRL寄存器（第10.6节）定义的访问控制列表执行如下权限控制：

- 允许访问该端口的实体列表：核0，核1，DMA，调试器
- 允许访问该端口的安全状态列表：安全/非安全与特权/非特权四种组合。

任何未通过上述任一检查的访问请求均被阻止访问下游端口，并在上游返回总线错误。

存在三处例外未启用总线安全过滤器，因其内部实现了自身的安全过滤机制：

- ACCESSCTRL模块本身，始终对所有主体可读，但在安全与特权写入操作上设有限制。
- 启动只读存储器，仅支持硬连线的安全访问。
- 单周期IO子系统（SIO），内部按安全和非安全区域分组

Cortex-M专用外设总线（PPB）寄存器同样不具备ACCESSCTRL权限，因为它们是处理器内部资源，非通过系统总线访问。PPB寄存器内部按安全和非安全区域分组。

2.1.3. 原子寄存器访问

每个外设寄存器块分配有4 KB的地址空间，寄存器通过以下四种方法之一访问，由地址译码选择。

- `Addr + 0x0000` : 正常读写访问
- `Addr + 0x1000` : 写操作时的原子异或 (XOR)
- `Addr + 0x2000` : 写操作时的原子位掩码置位
- `Addr + 0x3000` : 写操作时的原子位掩码清除

此机制使软件能够修改控制寄存器的单个字段，无需执行读-改-写操作。

外设自身则会在原位 (in-place) 修改其内容。若无此项功能，在中断服务例程与前台代码并发执行，或双处理器并行运行代码时，安全访问IO寄存器将极其困难。

四个原子访问别名共占用16 kB。原子写操作所需的时钟周期与普通写操作相同。RP2350上的大多数外设原生支持此功能，但某些外设 (I2C、UART、SPI和SSI) 通过总线中介器实现该功能。总线中介器将上游的原子写操作转换为外设边界处的下游读-改-写序列，代价是增加额外的时钟周期。

使用总线中介器的原子写操作相比普通写操作多消耗两个时钟周期。

以下寄存器不支持原子寄存器访问：

- SIO (第3.1节)，但某些单独寄存器 (例如GPIO) 具有置位、清除和异或别名。
- 通过自托管CoreSight窗口访问的任何寄存器，包括Arm Mem-AP和RISC-V调试模块。
- Cortex-M33专用外设总线 (PPB) 上的标准Arm控制寄存器，Raspberry Pi特定的EPPB寄存器除外。
- 通过SBPI桥访问的一次性编程寄存器。

2.1.4. APB桥接

APB桥提供了高速主AHB5互连与低带宽外设之间的接口。与在所有区域提供零等待状态访问的AHB5结构不同，APB访问读取操作至少需三个周期，写入操作至少需四个周期。

因此，APB部分总线结构的吞吐量低于AHB5部分。然而，其带宽足以饱和APB串行外设。

以下APB端口包含异步总线交叉，该交叉在APB桥的典型读写延迟基础上插入额外的停顿周期：

- ADC
- HSTX_CTRL
- OTP
- POWMAN

APB桥对下游传输的停滞实现固定超时机制。下游总线可能会无限期停滞，例如目标时钟停止时访问异步总线交叉，或通过自托管调试窗口中的Mem-AP访问系统APB寄存器时发生死锁 (详见第3.5.6节)。

当APB传输超过65,535个周期时，APB桥会放弃该传输并返回总线故障。此举保持系统总线可用，以便软件或调试器能够诊断传输时间过长的原因。

2.1.5. 窄带IO寄存器写入

RP2350上大多数内存映射I/O寄存器忽略总线读写访问的宽度。它们将所有写操作视为32位大小处理。这意味着软件无法使用字节或半字写操作来修改I/O寄存器的部分内容：任何写入地址，其30位最高有效位与寄存器地址匹配，都会影响整个寄存器的内容。

在不进行读-改-写序列的情况下，更新I/O寄存器部分内容的最佳方案是RP2350上的原子设定/清除/XOR操作（参见第2.1.3节）。此方法比字节或半字写操作更为灵活，因其可在单次操作中更新任意组合的字段。

进行8位或16位写操作（如Cortex-M33上的 `strb` 指令）时，窄值会在32位数据总线上被多次复制，广播到目标寄存器的所有8位或16位字段：

Pico示例：https://github.com/raspberrypi/pico-examples/blob/master/system/narrow_io_write/narrow_io_write.c 第19至62行

```

19 int main() {
20     stdio_init_all();
21
22     // 我们将使用 WATCHDOG_SCRATCH0 作为一个便于操作的32位读写寄存器
23     // 可向其分配任意值
24     io_rw_32 *scratch32 = &watchdog_hw->scratch[0];
25     // 将 scratch 寄存器在偏移 +0x0 和 +0x2 处别名为两个半字
26     volatile uint16_t *scratch16 = (volatile uint16_t *) scratch32;
27     // 将 scratch 寄存器在偏移 +0x0、+0x1、+0x2 和 +0x3 处别名为四个字节：
28     volatile uint8_t *scratch8 = (volatile uint8_t *) scratch32;
29
30     // 演示我们可以像正常一样读写 scratch 寄存器：
31     printf("Writing 32 bit value\n");
32     *scratch32 = 0xdeadbeef;
33     printf("Should be 0xdeadbeef: 0x%08x\n", *scratch32);
34
35     // 我们可以正常进行窄宽度读取——IO寄存器将此视为一次32位
36     // 读取，处理器/DMA将根据传输大小和地址最低有效位
37     // 选择正确的字节线
38     printf("\n逐字节读取\n");
39     // 小端序！
40     printf("应该是 ef be ad de : %02x ", scratch8[0]);
41     printf("%02x ", scratch8[1]);
42     printf("%02x ", scratch8[2]);
43     printf("%02x\n", scratch8[3]);
44
45     // 字节写入会在32位总线上复制四次，IO
46     // 寄存器通常会采样整个写入总线。
47     printf("\n在偏移量0写入8位值0xa5\n");
48     scratch8[0] = 0xa5;
49     // 一次性读取整个scratch寄存器的全部内容
50     printf("应为0xa5a5a5a5: 0x%08x\n", *scratch32);
51
52     // IO寄存器忽略地址最低有效位[1:0]及传输
53     // 大小，因此使用任何字节偏移量均无影响
54     printf("\n在偏移量1写入8位值\n");
55     scratch8[1] = 0x3c;
56     printf("应为0x3c3c3c3c: 0x%08x\n", *scratch32);
57
58     // 半字写入同样会在写数据总线上被复制
59     printf("\n在偏移量0写入16位值\n");
60     scratch16[0] = 0xf00d;
61     printf("应为0xf00df00d: 0x%08x\n", *scratch32);
62 }
```

要在RP2350上禁用此行为，请通过访问偏移量为 `+0x4000` 的外设，将地址的第 14位设置为1。此操作

会导致无效字节通道输出为零，而非输出复制的数据。在某些情况下，例如向PWM外设传输8位值的DMA，默认的复制行为并不理想。

2.1.6. 全局独占监控器

全局排他监视器使标准Arm和RISC-V原子指令能够安全地从两个核访问SRAM中的共享变量。这为操作共享变量的软件库提供支持，例如C11中的`stdatomic.h`头文件。

有关该监视器操作的详细规则，请参阅Armv8-M架构参考手册。

Arm将排他监视器的交互描述为一个处理单元（PE），该单元执行一系列总线访问。就RP2350而言，这是以下三者中的一个AHB5管理器：核心0负载/存储、核心1负载/存储以及DMA写入。DMA本身不执行排他访问，但其写操作会针对任一处理器上的排他序列受到监控。处理器的调试器访问与非调试器访问之间不作区分。

监视器监控由DMA写入及处理器负载/存储端口发起的所有SRAM传输，特别关注以下两种传输类型：

- AHB5排他读取：Arm `ldrex*`指令，RISC-V `lr.w`指令，以及RISC-V AMO的读取阶段（RP2350上的Hazard3内核将AMO实现为一对排他读/写操作，直到写入成功为止进行重试）。
- AHB5 排他写入：Arm `strex*`指令，RISC-V `sc.w`指令，以及RISC-V AMO的写回阶段。

基于上述观察，监视器保证同一处理执行单元的原子读-改-写序列（由排他读取随后成功的排他写入组成）不得与另一处理执行单元对相同预留粒度的成功写入（无论是否排他）交错进行。保留粒度是任何16字节的自然对齐SRAM区域。

当且仅当以下所有条件均满足时，排他写入才会成功：

- 该写入之前由相同PE执行了排他读取
- 自该排他读取以来，该PE未执行任何其他排他写入
- 该排他读取针对相同的保留粒度
- 该排他读取的大小相同（字节／半字／字）
- 该排他读取来自相同的安全和特权状态
- 自该排他读取以来，其他PE未对该粒度成功写入

如果上述条件未满足，全球排他监视器将在SRAM提交写入数据之前阻止该排他写入。该失败会报告给发起PE，例如通过Arm `strex`指令的非零返回值。

此实现的Armv8-M全局排他监视器也满足RISC-V `lr/sc`和 `amo*`指令的要求，但需注意不支持`RsrvEventual/PMA`。（实际上，尽管很容易构造诸如DMA在每个周期写入共享变量的刻意饥饿例子，但有界的LR/SC和AMO序列通常能够快速完成。）

⚠ 警告

安全软件应避免在非安全可访问内存中使用共享变量。此类变量容易受到通过反复执行非独占写入而导致的独占访问故意饿死攻击。

独占访问仅支持SRAM。系统将其他内存区域的独占访问视为普通读写操作，并通过例如Arm `strex`指令的非零返回值向发起处理单元报告独占失败。

2.1.6.1. 实现定义的监视器行为

Armv8-M架构参考手册对全局独占监视器的若干方面留待具体实现确定。为完整性起见，RP2350实现定义如下：

- 保留粒度大小固定为16字节
- 每个处理单元仅跟踪单个保留
- Arm `clrex`指令不影响全局监视器状态
- 任何处理单元（PE）执行的独占写操作都会清除该处理单元的全局保留
- 处理单元（PE）执行的非独占写操作，无论地址为何，均不会清除该处理单元的全局保留

仅以下情况会更新处理单元的保留标签，将其保留状态设置为 **Exclusive**：

- 对SRAM的独占读取

仅以下情况会将处理单元的保留状态从 **Exclusive** 变更为 **Open**：

- 另一处理单元对该处理单元保留执行的成功独占写操作
- 另一处理单元对该处理单元保留执行的非独占写操作
- 该处理单元执行的任何独占写操作
- 该处理单元的独占读取操作，非针对SRAM

保留粒度可能跨越多个SRAM银行，因此针对同一保留粒度的多个操作可能在同一周期完成。这可能导致以下问题情况：

- 对同一保留粒度进行多重独占写操作，且该粒度在各处理单元均已保留：此时编号最低的处理单元成功（顺序为 DMA < 核心0 < 核心1），其余处理单元操作失败。
- 在同一时钟周期内对同一保留单元进行非排他性与排他性写入的混合操作：在此情况下，排他性写入失败。
- 一个处理单元 *x*可以在另一处理单元 *y*尝试通过排他性加载保留相同保留单元的同一时钟周期内，向该保留单元写入数据：在此情况下，*y*的保留请求被授予（即该写入在逻辑上早于加载操作）。
- 一个处理单元 *x*可以在另一处理单元 *y*于同一时钟周期对不同保留单元进行新保留的同时，写入 *y*已保留的保留单元：在此情况下，同样授予*y*的保留请求。

这些规则可归纳为对同一时钟周期内可能发生在同一保留单元上的所有事件的逻辑排序：先以任意顺序处理所有普通写入，然后按处理单元编号递增顺序（DMA、核心0、核心1）处理所有排他性写入，最后以任意顺序处理所有加载操作。

2.1.6.2. 不支持排他性的区域

全局排他监视器仅支持特定地址范围内的排他事务。主系统SRAM在其整个地址范围内支持排他事务：`0x20000000`直到`0x20082000`。在支持排他事务的地址范围内，全局排他监视器：

- 跟踪所有参与处理单元（PE）之间的排他序列。
- 根据观察到的顺序准确驱动排他成功或失败响应。
- 取消失败的排他写操作，确保其不产生任何影响。

排他事务不支持该范围之外；所有排他访问（包括排他读和排他写）均报告排他失败，且排他写操作不被抑制。

在无排他事务支持的区域外，负载/存储排他循环将无限执行，同时仍会影响SRAM内容。该规则适用于执行排他读/写的Arm处理器以及执行`lr.w`/`sc.w`指令的RISC-V处理器。然而，在Hazard3处理器上执行`amo*.w`指令将触发存储/原子内存操作故障，因为硬件

检测到排他读失败并终止操作，以避免无限循环。

建议不要对主SRAM以外的区域进行独占访问。主SRAM以外的共享变量可通过主SRAM中的锁变量、SIO自旋锁，或无需独占访问的锁定协议（例如无锁队列）加以保护。

2.1.7. 总线性能计数器

总线性能计数器自动统计对主AHB5交叉开关仲裁器的访问次数。这些计数器有助于诊断高流量下的性能问题。

共有四个性能计数器，编号从PERFCTR0开始。每个计数器为24位饱和计数器。计数器数值可从BUSCTRL_PERFCTR x 读取，向BUSCTRL_PERFCTR x 写入任意值可将其清零。每个计数器每次可计数20个可用事件中的一个，通过BUSCTRL_PERFSEL x 进行选择。更多信息详见第12.15.4节。

2.2. 地址映射

设备的地址映射如表8所示被划分为若干部分。详细信息请参见下述章节。

访问未映射的地址范围时将引发总线错误。

表8左侧列中的每个链接均指向对应地址范围的详细地址映射。详细地址映射内的每个地址均提供指向相关文档的链接。

先对地址的第31至28位进行粗略地址解码：

表8. 地址映射概要

总线分段	基地址
只读存储器	0x00000000
XIP	0x10000000
SRAM	0x20000000
APB 外设	0x40000000
AHB 外设	0x50000000
核心本地外设 (SIO)	0xd0000000
Cortex-M33 私有寄存器	0xe0000000

2.2.1. 只读存储器 (ROM)

只读存储器可被DMA、处理器加载/存储及指令抓取访问。其位于地址0，这是设备复位时Arm处理器的起始点。

表9。只读存储器总线段地址映射

总线端点	基地址
ROM_BASE	0x00000000

2.2.2. 执行代码于闪存 (XIP)

XIP可被DMA、处理器加载/存储及指令抓取访问。此地址范围包含映射至外部存储设备的64MB空间的多个镜像。在RP2350上，低32MB由QSPI存储接口 (QMI) 占用，剩余部分为保留。QMI控制寄存器位于APB寄存器区。

表10。XIP总线段地址映射

总线端点	基地址
XIP_BASE	0x1000000
XIP_NOCACHE_NOALLOC_BASE	0x1400000
XIP_MAINTENANCE_BASE	0x1800000
XIP_NOCACHE_NOALLOC_NOTRANSFER_BASE	0x1c00000

i 注意

XIP_SRAM_BASE不再作为独立地址范围存在。缓存即SRAM的功能现通过在缓存的XIP地址空间内固定缓存行来实现。

2.2.3. 静态随机存取存储器 (SRAM)

SRAM可供DMA、处理器加载/存储操作及处理器指令获取访问。

SRAM0-3与SRAM4-7始终按地址的3:2位进行条带分布：

表11。SRAM总线段映射，SRA M0-7(条带式)

总线端点	基地址
SRAM_BASE	0x2000000
SRAM_STRIPED_BASE	0x2000000
SRAM0_BASE	0x2000000
SRAM4_BASE	0x2004000
SRAM_STRIPED_END	0x2008000

共有两个条带区域，每个大小为256 kB，且均跨越4个SRAM银行。SRAM0-3属于SRAM0电源域，SRAM4-7属于SRA M1电源域。

SRAM8-9始终为非条带式：

表12。SRAM总线段映射，SRA M8-9(非条带式)

总线端点	基地址
SRAM8_BASE	0x2008000
SRAM9_BASE	0x2008100
SRAM_END	0x2008200

这些较小的SRAM块适用于承载高带宽数据结构，如处理器堆栈。它们属于SRAM1电源域。

2.2.4. APB 寄存器

APB外设寄存器仅允许处理器负载/存储操作及DMA访问。指令取指操作将始终失败。

APB外设段提供对控制寄存器和配置寄存器的访问，以及对低带宽外设的数据访问。APB写入操作至少需四个周期，AP B读取操作至少需三个周期。

表13。APB总线段地址映射

总线端点	基地址
SYSINFO_BASE	0x4000000
SYSCFG_BASE	0x40008000

总线端点	基地址
CLOCKS_BASE	0x40010000
PSM_BASE	0x40018000
RESETS_BASE	0x40020000
IO_BANK0_BASE	0x40028000
IO_QSPI_BASE	0x40030000
PADS_BANK0_BASE	0x40038000
PADS_QSPI_BASE	0x40040000
XOSC_BASE	0x40048000
PLL_SYS_BASE	0x40050000
PLL_USB_BASE	0x40058000
ACCESSCTRL_BASE	0x40060000
BUSCTRL_BASE	0x40068000
UART0_BASE	0x40070000
UART1_BASE	0x40078000
SPI0_BASE	0x40080000
SPI1_BASE	0x40088000
I2C0_BASE	0x40090000
I2C1_BASE	0x40098000
ADC_BASE	0x400a0000
PWM_BASE	0x400a8000
TIMER0_BASE	0x400b0000
TIMER1_BASE	0x400b8000
HSTX_CTRL_BASE	0x400c0000
XIP_CTRL_BASE	0x400c8000
XIP_QMI_BASE	0x400d0000
WATCHDOG_BASE	0x400d8000
BOOTRAM_BASE	0x400e0000
ROSC_BASE	0x400e8000
TRNG_BASE	0x400f0000
SHA256_BASE	0x400f8000
POWMAN_BASE	0x40100000
TICKS_BASE	0x40108000
OTP_BASE	0x40120000
OTP_DATA_BASE	0x40130000
OTP_DATA_RAW_BASE	0x40134000
OTP_DATA_GUARDED_BASE	0x40138000

总线端点	基地址
OTP_DATA_RAW_GUARDED_BASE	0x4013c000
CORESIGHT_PERIPH_BASE	0x40140000
CORESIGHT_ROMTABLE_BASE	0x40140000
CORESIGHT_AHB_AP_CORE0_BASE	0x40142000
CORESIGHT_AHB_AP_CORE1_BASE	0x40144000
CORESIGHT_TIMESTAMP_GEN_BASE	0x40146000
CORESIGHT_ATB_FUNNEL_BASE	0x40147000
CORESIGHT_TPIU_BASE	0x40148000
CORESIGHT_CTLT_BASE	0x40149000
CORESIGHT_APB_AP_RISCV_BASE	0x4014a000
GLITCH_DETECTOR_BASE	0x40158000
TBMAN_BASE	0x40160000

2.2.5. AHB 寄存器

AHB 外设寄存器仅允许处理器加载/存储和 DMA 访问。指令取指操作将始终失败。

AHB 外设段提供对高带宽外设的访问。最小读写代价为一个周期，外设可能插入最多一个等待状态。

表 14。 AHB 外设总线段地址映射

总线端点	基地址
DMA_BASE	0x50000000
USBCTRL_BASE	0x50100000
USBCTRL_DPRAM_BASE	0x50100000
USBCTRL_REGS_BASE	0x50110000
PIO0_BASE	0x50200000
PIO1_BASE	0x50300000
PIO2_BASE	0x50400000
XIP_AUX_BASE	0x50500000
HSTX_FIFO_BASE	0x50600000
CORESIGHT_TRACE_BASE	0x50700000

2.2.6. 核心本地外设 (SIO)

SIO 仅允许处理器加载/存储访问。包含需要两个核心同时单周期访问的寄存器，如 GPIO 寄存器。访问始终为零等待状态。

表 15。SIO总线段地址映射

总线端点	基地址
SIO_BASE	0xd0000000
SIO_NONSEC_BASE	0xd0020000

2.2.7. Cortex-M33 私有外设

PPB 仅对处理器的加载/存储操作开放访问。

PPB 区域包含 Arm 定义的标准控制寄存器、部分这些寄存器的非安全别名及 Raspberry Pi 定义的少量其他核心本地寄存器（EPPB）。

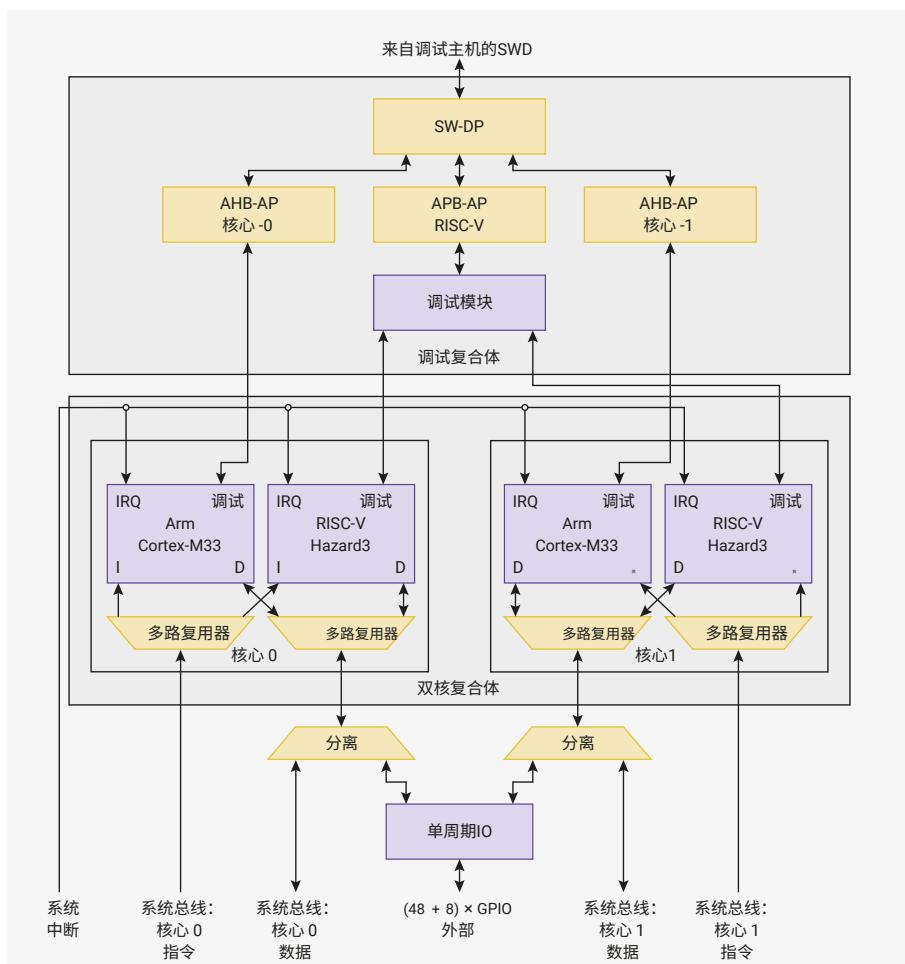
这些地址仅限 Arm 处理器访问：RISC-V 处理器访问将导致总线错误。

表 16。PPB总线段地址映射

总线端点	基地址
PPB_BASE	0xe0000000
PPB_NONSEC_BASE	0xe0020000
EPPB_BASE	0xe0080000

第3章 处理器子系统

图6. RP2350 处理器子系统将两个处理器连接至系统总线、外设中断、GPIO 以及来自外部调试主机的串行线调试 (SWD) 连接。它还包含紧密耦合的外设及用于同步和通信的外设，统称为单周期 I/O 子系统 (SIO)。



RP2350 是对称双核系统。两个核心同时且独立运行，提供高处理吞吐量，并能将中断路由至不同核心，以提升中断处理的吞吐量和延迟。两个核心对系统总线具有对称视图；RP2350 上所有内存资源在两个核心上均可平等访问，性能一致。

每个核心配备一对 32 位 AHB5 链接至系统总线。一个链路专用于指令获取，另一个专用于加载或存储指令及调试器访问。每个核心每周期可执行一次指令获取及一次加载或存储访问，前提是下游总线端口无冲突。

系统总线上有两个核心插槽，分别称为**core 0** 和 **core 1**，在本数据手册中采用此命名。

(在寄存器文档中，它们亦可同义称为 core0、core1、proc0 和 proc1。) 各插槽所插入的处理器可在启动时选择：

- 一颗实现 Armv8-M Main 指令集及其扩展的 Cortex-M33 处理器
- 一颗实现 RV32IMAC 指令集及其扩展的 Hazard3 处理器

默认选用 Cortex-M33。未使用的处理器将被复位，并在顶层对其时钟进行门控。

未使用的处理器不消耗动态功率。有关架构选择硬件的详细信息，请参见第 3.9 节。

两个 Cortex-M33 实例完全相同。它们配置了安全、DSP 及 FPU 扩展，以及

8x SAU 区域、8 个安全 MPU 区域和 8 个非安全 MPU 区域。第 3.7 节对 Cortex-M33 处理器及其在 RP2350 上的具体配置进行了说明。两个 Hazard3 实例也彼此完全相同；有关 Hazard3 处理器功能和操作的内容，请参见第 3.8 节。

Cortex-M33对Armv8-M安全扩展（亦称TrustZone-M）的实现，将设备上运行的受信任软件与非受信任软件隔离。RP2350在整个系统中扩展了Arm安全态与非安全态的严格分区功能，包括为每个安全域分配外设、GPIO和DMA通道的能力。

有关RP2350安全架构中Armv8-M安全扩展特性的概览，请参见第10.2节。

图6未显示Cortex-M33的协处理器。该协处理器与核心紧密耦合，提供每个时钟周期64位的数据向Arm寄存器文件的传输速率。您可将其视为位于示意图中Cortex-M33模块内部。RP2350为每个Cortex-M33配备了以下协处理器：

- 协处理器 0：GPIO 协处理器（GPIOC），详见第3.6.1节
- 协处理器 4和 5：双精度协处理器（DCP）的安全与非安全实例，详见第3.6.2节
- 协处理器 7：冗余协处理器（RCP），详见第3.6.3节

外部调试主机可通过串行线调试（SWD）总线访问两个核心。主机能够：

- 运行、停止及重置核心
- 检查内部核心状态，如寄存器
- 从核心视角访问内存
- 将代码加载至设备并执行

[第3.5节介绍了调试硬件，及Arm处理器上可用的指令追踪硬件。](#)

系统中的外设会断言**中断请求**（IRQ），以请求处理器响应。例如，当UART外设接收到字符时，会断言其中断请求，以使处理器能够从接收FIFO中读取字符。所有中断信号均路由至两个核心，核心内部的中断控制器选择其所需订阅的中断信号。第3.2节定义了系统级IRQ编号及Arm不可屏蔽中断（NMI）的详细信息。

第3.3节中描述的事件信号机制使处理器在等待系统中其他处理器完成任务或释放资源时进入休眠状态。每个处理器均能检测到由另一处理器发出的事件。

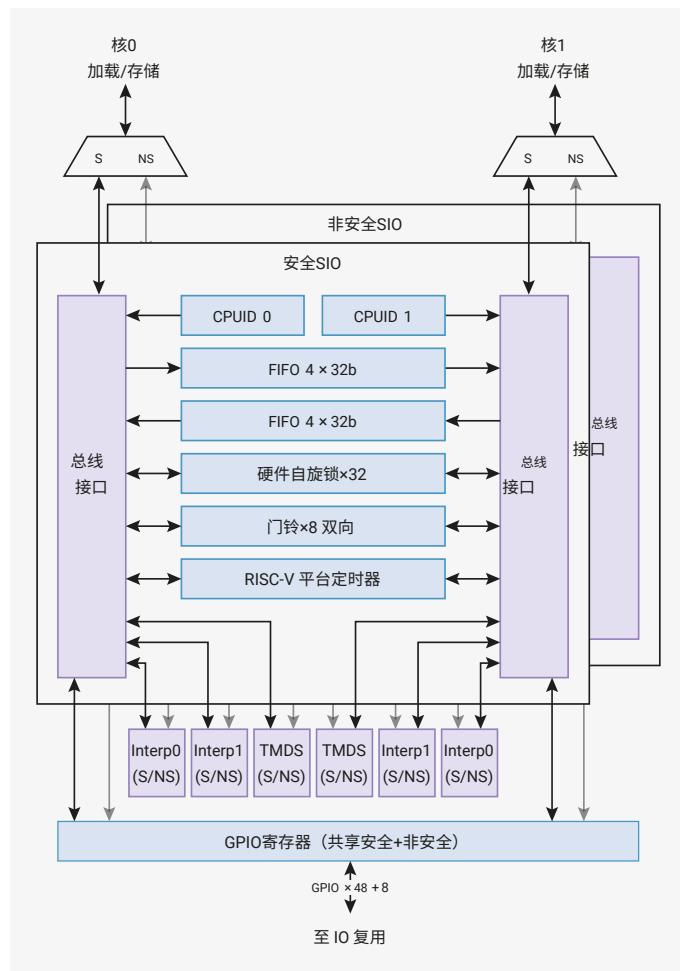
处理器还会检测到由第2.1.6节所述的全局排他监控器生成的排他事件，该硬件模块允许处理器通过原子读-改-写序列安全操作共享变量。

3.1. SIO

单周期IO子系统（SIO）包含需要处理器低延迟且确定性访问的外设。该子系统通过AHB结构进行访问。SIO为每个处理器配备了专用总线接口，如图7所示。

图7. 单周期 I/O
模块包含处理器必须快速访问的寄存器。
FIFO、门铃和自旋锁支持两个核心之间的消息传递与同步。共享 GPIO 寄存器提供对具备 GPIO 功能引脚的快速直接访问。

插值器能够加速常见的软件任务。大多数 SIO 硬件针对安全与非安全访问进行了分组（复制）。灰色箭头表示非安全访问的总线连接。



SIO 包含：

- CPUID 寄存器，核心 0/1 上读取值分别为 0/1（见第3.1.2节）
- 用于在核心之间传递有序消息的邮箱FIFO（第3.1.5节）
- 用于在累计和无序事件上中断对方核心的门铃（第3.1.6节）
- 用于实现临界区的硬件自旋锁，无需使用独占总线访问（第3.1.4节）
- 插值器（第3.1.10节）和TMDS编码器（第3.1.9节）
- 标准RISC-V 64位平台定时器（第3.1.8节），可被Arm与RISC-V软件共用
- 用于快速软件位段操作的GPIO寄存器（第3.1.3节），由两个核心共享访问

大多数SIO硬件均为安全和非安全访问提供了重复设计。非安全访问FIFO寄存器时，观察到的FIFO在物理上与安全访问相同地址的FIFO不同，确保安全与非安全软件的消息不混合：第3.1.1节对此安全/非安全分区做了详细说明。

3.1.1. 安全与非安全 SIO

为了实现安全与非安全软件的隔离，同时保持在任一域中运行的软件编程模型的一致性，SIO被复制为安全与非安全两套。大多数硬件均在这两套之间复制，包括：

- 邮箱FIFO
- 门铃寄存器

- 向处理器输出的中断信号
- 自旋锁

例如，核心0上的非安全代码可以通过非安全实例的邮箱FIFO向核心1上的非安全代码传递消息。此消息将触发一个非安全中断，该中断信号独立于安全FIFO的中断线。这不会干扰可能同时进行的任何安全消息传递，且非安全代码无法监听安全消息，因为其无权访问安全邮箱。

在安全域与非安全域中运行的软件可以相同，处理器对SIO的总线访问将自动路由至安全版或非安全版的邮箱寄存器。

以下硬件未被重复：

- GPIO寄存器为共享资源，非安全访问依据ACCESSCTRL寄存器中的GPIO_NSmask0和GPIO_NSmask1中定义的非安全GPIO掩码在每个GPIO级别进行过滤
- RISC-V标准平台计时器（MTIME, MTIMEH），亦可被Arm处理器使用，仅存在于安全SIO中，因其为RISC-V的机器模式外设
- 内插器和TMDS编码器外设可通过PERI_NONSEC寄存器分配至安全或非安全SIO

对起始于 `0xd0000000` (SIO_BASE) 地址范围内SIO寄存器的访问，将映射至与总线访问安全属性匹配的SIO存储器块此即表示，来自Arm安全态或RISC-V机器模式的访问将访问安全SIO存储器块，来自Arm非安全态或RISC-V用户模式的访问将访问非安全SIO存储器块

此外，安全访问可使用从 `0xd0020000` (SIO_NONSEC_BASE) 起的镜像地址范围，以访问SIO的非安全视图，例如，利用非安全门铃中断在另一核上运行的非安全代码。非安全代码尝试访问该地址范围将引发总线错误。

ⓘ 注意

Secure-to-Non-secure镜像的 `0x20000` 偏移与PPB镜像在 `0xe0000000` (PPB_BASE) 及 `0xe0020000` (PPB_NONSEC_BASE) 处相匹配，功能类似。

ⓘ 注意

调试访问依据调试器总线访问的安全属性映射到安全/非安全SIO，该安全属性可能与核心停下时的安全状态不同。

3.1.2. CPUID

当核心0读取CPUID SIO寄存器时返回值为0，核心1读取时返回值为1。此功能有助于软件识别正在运行当前应用程序的核心。初始启动序列同样依赖此检查：两个内核同时运行，内核1进入深度休眠状态，内核0继续执行主启动序列。

! 重要

请勿将SIO CPUID寄存器与每个处理器内部私有外围总线上的Cortex-M33 CPUID寄存器混淆，后者列明处理器的部件编号及版本。

注意

读取每个Hazard3内核的MHARTID CSR返回与CPUID相同的值：内核0为0，内核1为1。

3.1.3. GPIO 控制

SIO GPIO寄存器控制已选择SIO功能（功能 5）的GPIO。该功能支持以下引脚：

- 所有用户GPIO（GPIO 0至29，或根据封装选项为0至47）
- QSPI引脚
- USB DP/DM引脚

所有SIO GPIO控制寄存器均成对出现。每对寄存器中地址较低者（例如GPIO_IN）连接GPIO 0至31，地址较高者（例如GPIO_HI_IN）连接GPIO 32至47、QSPI引脚及USB DP/DM引脚。

注意

要通过SIO的GPIO寄存器驱动引脚，必须首先将该引脚的GPIO多路复用器配置为选择SIO GPIO功能。详见表646。

这些GPIO寄存器由两个核心共享，两者可同时访问。寄存器分为三组：

- 输出寄存器GPIO_OUT和GPIO_HI_OUT用于设置GPIO输出电平，0表示低电平，1表示高电平。
- 输出使能寄存器GPIO_OE和GPIO_HI_OE用于启用输出驱动，0表示高阻，1根据GPIO_OUT和GPIO_HI_OUT驱动高电平或低电平。
- 输入寄存器GPIO_IN和GPIO_HI_IN使处理器能够采样GPIO的当前状态。

读取GPIO_IN可一次返回最多32个输入值，软件随后对感兴趣的单个引脚进行掩码处理。

SDK: https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h 第869至879行

```

869 static inline bool gpio_get(uint gpio) {
870 #ifdef NUM_BANK0_GPIOS <= 32
871     return sio_hw->gpio_in & (1u << gpio);
872 #else
873     if (gpio < 32) {
874         return sio_hw->gpio_in & (1u << gpio);
875     } else {
876         return sio_hw->gpio_hi_in & (1u << (gpio - 32));
877     }
878 #endif
879 }
```

OUT 和 OE 寄存器还具有原子操作的 SET、CLR 和 XOR 别名。这使软件能够在一次操作中更新部分引脚。这保证了 GPIO 访问的并发安全，无论是在双核之间，还是单核的中断处理程序与前台代码之间。

SDK: https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h 第 918 - 924 行

```

918 static inline void gpio_set_mask(uint32_t mask) {
919 #ifdef PICO_USE_GPIO_COPROCESSOR
920     gpioc_lo_out_set(mask);
```

```

921 #else
922     sio_hw->gpio_set = mask;
923 #endif
924 }

```

SDK: https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h 行 965 - 971

```

965 static inline void gpio_clr_mask(uint32_t mask) {
966 #ifdef PICO_USE_GPIO_COPROCESSOR
967     gpioc_lo_out_clr(mask);
968 #else
969     sio_hw->gpio_clr = mask;
970 #endif
971 }

```

SDK: https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h 行 1155 - 1180

```

1155 static inline void gpio_put(uint gpio, bool value) {
1156 #ifdef PICO_USE_GPIO_COPROCESSOR
1157     gpioc_bit_out_put(gpio, value);
1158 #elif NUM_BANK0_GPIOS <= 32
1159     uint32_t mask = 1ul << gpio;
1160     if (value)
1161         gpio_set_mask(mask);
1162     else
1163         gpio_clr_mask(mask);
1164 #else
1165     uint32_t mask = 1ul << (gpio & 0x1fu);
1166     if (gpio < 32) {
1167         if (value) {
1168             sio_hw->gpio_set = mask;
1169         } else {
1170             sio_hw->gpio_clr = mask;
1171         }
1172     } else {
1173         if (value) {
1174             sio_hw->gpio_hi_set = mask;
1175         } else {
1176             sio_hw->gpio_hi_clr = mask;
1177         }
1178     }
1179 #endif
1180 }

```

如果两个处理器在同一时钟周期内同时写入同一个 OUT或 OE寄存器（或其任何 SET/CLR/XOR别名），其结果相当于先由核心0写入，然后核心1紧接着写入。例如，如果核心0在同一时钟周期内对某个位执行 SET 操作，而核心1对其执行 XOR 操作，则该位最终值为 0。

i 注意

这是一个概念性模型，用以说明两个核心同时写入 GPIO 寄存器时产生的结果。该寄存器在任何时刻均不会包含中间值。在前例中，若引脚初始值为 0，且核心 0 执行 SET 操作而核心 1 执行 XOR 操作，则整个时钟周期内 GPIO 输出保持低电平。

GPIO 寄存器不仅在核心间共享，也在安全域间共享。安全与非安全 SIO 提供了同一 GPIO 寄存器的不同视图，该寄存器始终映射为 GPIO 功能 5。

然而，非安全 SIO 仅能访问由 ACCESSCTRL 寄存器 GPIO_NSMASK0 和 GPIO_NSMASK1 配置的 GPIO 非安全掩码中启用的引脚。NSMASK 寄存器的布局与 SIO 寄存器一致——例如，**QSPI_SCK** 在 GPIO_HI_IN 和 GPIO_NSMASK1 中均位于第 26 位。

当引脚未在非安全代码中启用时：

- 从非安全上下文对相应 GPIO 寄存器的写入操作无效。
- 从非安全上下文读取的值均为零。
- 从安全上下文的读写操作正常执行，使用安全区域。

SIO 协处理器端口（第3.6.1节）为 Cortex-M33 处理器访问 SIO GPIO 寄存器提供了专用指令。其中包括在单次操作中读取和写入 64 位数据的能力。

3.1.4. 硬件自旋锁

SIO 提供 32 个硬件自旋锁，可用于管理对共享软件资源的互斥访问。每个自旋锁为一位标志，映射到不同地址（从 SPINLOCK0 至 SPINLOCK31）。软件通过以下操作之一与自旋锁交互：

- 读取：尝试获取锁。若成功获取锁，则读取值非零；若锁已被先前读取获取，则读取值为零。
- 写入（任意值）：释放锁。下一次尝试获取该锁将成功。

若两个内核在同一时钟周期尝试获取相同锁，内核 0 将成功。

通常，软件通过反复轮询锁位（“自旋”锁）来获取锁，直至成功。若锁长时间被持有，此方式效率低，故自旋锁通常用于保护更高级原语（如互斥锁、信号量及队列）中的短关键区段。

出于调试目的，可通过 SPINLOCK_ST 观察所有 32 个自旋锁的当前状态。

i 注意

RP2350 对安全与非安全 SIO 寄存器组设有独立自旋锁，因共享寄存器将允许非安全代码故意阻塞获取锁的安全代码。详见第3.1.1节。

i 注意

RP2350 处理器支持标准原子/独占访问指令，结合全局独占监视器（第2.1.6节），使两个内核安全共享 SRAM 变量。SIO 自旋锁仍保留以兼容 RP2040。

i 注意

由于RP2350-E2，对偏移量 `+0x180`以上的新SIO寄存器的写入操作会产生别名，影响自旋锁，导致伪锁释放。SDK默认使用原子内存访问来实现`hardware_sync_spin_lock`API，此为RP2350 A2上的一种权宜之计。

3.1.5. 处理器间 FIFO（邮箱）

SIO包含两个FIFO队列，用以在两个核心之间传递数据、消息或有序事件。每个FIFO宽度为32位，深度为四个条目。其中一个FIFO仅允许核心0写入，核心1读取。另一个FIFO仅允许核心1写入，核心0读取。

每个核心通过写入FIFO_WR来向其输出FIFO写入数据，通过读取FIFO_RD来从其输入FIFO读取数据。
状态寄存器FIFO_ST提供下列状态信号：

- 输入FIFO中含有数据（`VLD`）。
- 输出FIFO还有空间可供写入（`RDY`）。
- 输入FIFO曾在空时被读取过（`ROE`）。
- 过去某个时刻曾向已满的输出FIFO写入数据（`WOF`）。

向已满的输出FIFO写入数据或从空的输入FIFO读取数据均不影响FIFO状态。

FIFO当前的内容和容量水位保持不变。然而，这意味着访问FIFO的软件出现了数据丢失或接收无效数据，因此会触发一个粘滞错误标志（`ROE`或 `WOF`）。

SIO为每个核心提供了FIFO中断输出，用以通知核心已收到FIFO数据。这是一种核心本地中断，在每个核心上映射为相同的IRQ号（`SIO_IRQ_FIFO`，中断号 25）。非安全FIFO中断使用独立的中断线（`SIO_IRQ_FIFO_NS`，中断号 27）。无法对另一个核心的FIFO触发中断。

每个IRQ输出是该核心FIFO_ST寄存器中 `VLD`、`ROE`和 `WOF`位的逻辑或：只要这三个位中有任一位为高，IRQ即被置位；当这三个位全为低时，IRQ清除。清除 `ROE`和 `WOF`标志，需向`FIFO_ST`写入任意值；清除 `VLD`标志，需从FIFO读取数据直至其为空。

如果处理器中断控制器中对应的中断线被使能，则每当数据出现在其FIFO中，或者发生无效的FIFO操作（如空时读取、满时写入），处理器都会响应中断。

i 注意

`ROE`和 `WOF`仅在软件出现异常行为时才会被置位。通常，中断处理程序在FIFO出现数据时触发，并置位 `VLD`标志。随后，中断处理程序通过读取FIFO中的数据直到`VLD`再次清零，从而清除IRQ。

多处理器间FIFO和事件信号被引导只读存储器（第5章）中的`wait_for_vector`例程使用，核心1保持睡眠状态，直到被唤醒，并通过FIFO接收其初始栈指针、入口地址和向量表。

i 注意

RP2350为安全和非安全SIO银行分别设有独立的FIFO和中断。详见第3.1.1节。

3.1.6. 门铃

门铃寄存器会在对向核心引发中断。每个方向有8个门铃标志，汇总为每个核心的单一门铃中断。这是一个核心本地中断：每个核心上的相同中断号（`SIO_IRQ_BELL`，中断号 26）通知该核心收到门铃中断。

信箱FIFO用于跨核心事件，计数和顺序非常重要；而门铃用于累积性事件（即可能多次触发，但仅需响应一次），且可按任意顺序处理。

向DOORBELL_OUT_SET寄存器写入非零值将触发对端核心的门铃中断。中断将持续激活，直到所有相关位被清除。通常，对端核心进入门铃中断处理程序，读取其DOORBELL_IN_CLR寄存器以获取活动门铃标志的掩码，然后写回以确认并清除中断。

DOORBELL_IN_SET寄存器允许处理器触发自身的门铃中断。当响铃例程可安排在任一核心执行时，此方法非常有用。同理，为了对称，处理器可使用DOORBELL_OUT_CLR寄存器清除对方核心的门铃标志：这对初始化代码有用，但一般应避免使用，以防在确认针对对方核心的中断时发生竞态条件。

核心可随时读取其DOORBELL_OUT_SET或DOORBELL_OUT_CLR寄存器（两者返回相同结果），以查询发送至对方核心的门铃中断状态。同样，读取DOORBELL_IN_SET或DOORBELL_IN_CLR任一寄存器均可返回发送至本核心的门铃中断状态。

i 注意

RP2350 为安全与非安全 SIO 银行分别设置了独立的每核心门铃中断信号及门铃寄存器。非安全门铃通过[SIO_IRQ_BELL_NS](#)发送，中断编号为 28。详见第 3.1.1 节。

3.1.7. 整数除法器

RP2040 的内存映射整数除法外设在 RP2350 上不存在，原因是处理器支持除法指令。之前分配给除法寄存器的地址空间现已保留。

3.1.8. RISC-V 平台定时器

该64位定时器为RISC-V特权规范中描述的标准外设，RP2350上的Arm和RISC-V处理器均可使用。它触发每核[SIO_IRQ_MTIMECMP](#)系统级中断（见第3.2节），以及RISC-V处理器上的mip.mtip定时器中断。

存在一个64位计数器，两个核心共享使用。低半部和高半部可通过MTIME和MTIMEH SIO寄存器访问。请使用以下步骤，通过32位寄存器访问安全读取64位时间：

1. 读取高半部MTIMEH。
2. 读取低半部MTIME。
3. 再次读取高半部。
4. 若两次读取的高半部值不同，则循环重读。

该过程类似于读取RP2350系统定时器（见第12.8节）。循环重读仅在定时器恰好于32位溢出瞬间被读取时发生，且概率极低。如需恒定时间操作，当两次高半部读取值不同时，可选择将低半部清零。

定时器中断基于每核64位时间比较值产生，该值通过MTIMECMP和MTIMECMPPH SIO寄存器访问。每个核心均拥有这组寄存器的独立副本，且访问地址相同。当MTIME寄存器指示的当前时间大于或等于该核心的MTIMECMP时，便断言该核的中断。请按以下步骤写入新的64位定时器比较值，以避免产生虚假中断：

1. 向MTIMECMP写入全1（二进制，确保该值大于或等于旧值及目标值的低半部分）。
2. 向MTIMECMPPH写入目标值的高半部分（合并的64位值仍大于或等于目标值）。

3. 向MTIMECMP写入目标值的低半部分。

RISC-V定时器可以计数系统级节拍生成器产生的节拍（详见第8.5节）或系统时钟周期，该选择由MTIME_CTRL寄存器控制。采用1微秒时间基准，以保证与大多数RISC-V软件的兼容性。

3.1.9. TMDS 编码器

每个内核均实现了TMDS编码算法，该算法详述于DVI 1.0规范第3章。通常，HSTX外设（第12.11节）支持较低的处理器负载以实现DVI-D输出，并兼容更广泛的像素格式，但SIO TMDS编码器则用于不支持HSTX功能的GPIO。

TMDS_CTRL寄存器支持多种输入像素格式配置，范围涵盖16位RGB至1位单色。编码器配置完成后，处理器每次向TMDS_WDATA写入32位颜色数据，随后从输出寄存器读取TMDS数据符号。根据像素格式不同，每次写入TMDS_WDATA后可能需读取多个TMDS符号。无任何停顿：编码速率完全受限于处理器的加载/存储带宽，最高可达每核每时钟周期一次32位读写操作。

为了支持帧缓冲区/扫描缓冲区分分辨率低于显示分辨率，输出寄存器设有peek和pop别名（例如TMDS_PEEK_SINGLE和TMDS_POP_SINGLE）。读取任一寄存器将推进编码器的DC平衡计数器，但只有pop别名会移位TMDS_WDATA中的颜色数据，从而使得能通过同一输入像素生成多个DC平衡正确的TMDS符号。

TMDS编码器外设未在安全域间重复分配。其在复位时被分配至安全SIO，并可通过PERI_NONSEC寄存器重新分配至非安全SIO。

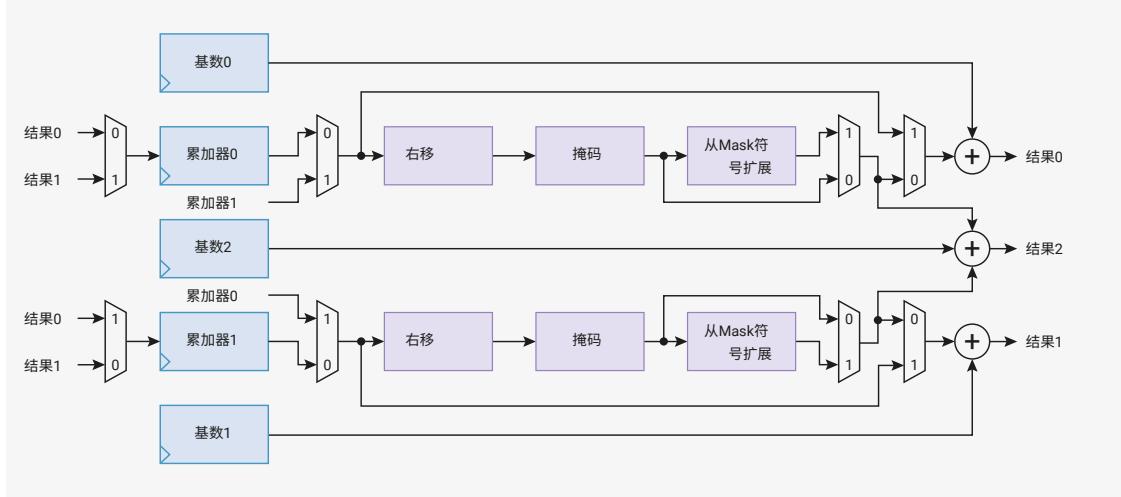
3.1.10. 插值器

每个核心配备两个插值器（**INTERP0**和**INTERP1**），能够通过将特定预配置操作合并为单个处理器周期来加速任务处理。插值器适用于预配置操作多次重复的场景，有助于在时间关键段落中减少CPU周期数及CPU寄存器使用量。

插值器已在SDK中加速音频操作。其灵活的配置使得优化许多其他任务成为可能，包括：

- 量化
- 抖动
- 表查地址生成
- 仿射纹理映射
- 解压缩
- 线性反馈

图8. 一个插值器。两个累加器寄存器和三个基寄存器允许处理器进行单周期的读写访问。插值器被组织为两个通道，分别对两个累加器执行屏蔽、移位和符号扩展操作。通过将中间的移位/屏蔽加到三个基寄存器上，从而产生三种可能的结果。从左至右，每个通道上的多路复用器由CTR L寄存器中的以下标志控制：



处理器可在一周期内读写任何插值器寄存器，结果将在下一周期准备就绪。处理器还可通过写入对应的ACCUMx_ADD寄存器，对两个累加器ACCUM0或ACCUM1之一执行加法操作。

这三种结果在只读寄存器PEEK0、PEEK1和PEEK2中可用。从这些位置读取不会改变插值器的状态。结果也别名于位置POP0、POP1、POP2；从POPx别名读取返回与对应的PEEKx相同的结果，同时将该通道结果写回累加器。每次读取结果时，使用POPx别名以推进插值器状态。

您可以通过以下操作模式调整插值器的行为：

- 两个数值之间的分数混合
- 将数值限定在指定范围内。

以下示例显示了一个简单的弹出通道结果以实现简单迭代反馈的例子。

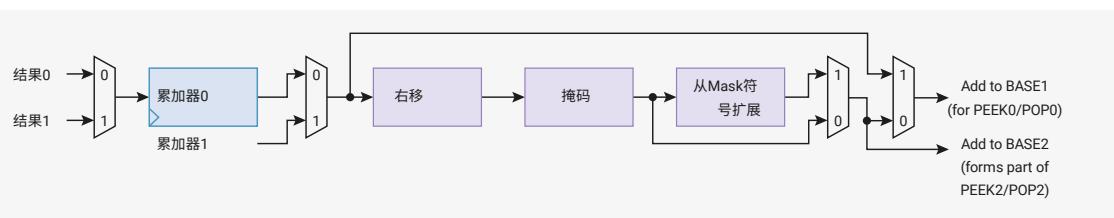
Pico 示例：https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c 第11至23行

```

11 void times_table() {
12     puts("9 的乘法表: ");
13
14     // 在此核心上初始化 interp0 的通道 0
15     interp_config cfg = interp_default_config();
16     interp_set_config(interp0, 0, &cfg);
17
18     interp0->accum[0] = 0;
19     interp0->base[0] = 9;
20
21     for (int i = 0; i < 10; ++i)
22         printf("%d\n", interp0->pop[0]);
23 }
```

3.1.10.1 通道操作

图 9. 每个插值器的每个通道均可配置为对某个累加器执行掩码、移位及符号扩展操作。该结果输入加法器生成最终输出，且可选择性地在每次读取时反馈至累加器。数据路径可通过若干32位多路复用器进行配置。从左到右，这些均由以下 CTRL 标志控制：
：



每个通道依次执行以下三项操作：

- 向右移位`CTRL_LANEx_SHIFT`位（范围0至31位）
- 掩码位于`CTRL_LANEx_MASK_LSB`至`CTRL_LANEx_MASK_MSB`（含）之间，均在第0至31位范围内
- 从掩码最高位进行符号扩展，即当设置`CTRL_LANEx_SIGNED`时，将`CTRL_LANEx_MASK_MSB`位扩展至所有更高有效位

例如，当：

- ```
CROSS_RESULT,
CROSS_INPUT, SIGNED,
以及 ADD_RAW。

```
- `ACCUM0 = 0xdeadbeef`
  - `CTRL_LANE0_SHIFT = 8`
  - `CTRL_LANE0_MASK_LSB = 4`
  - `CTRL_LANE0_MASK_MSB = 7`
  - `CTRL_SIGNED = 1`

则通道0在每阶段将产生如下结果：

- 右移8位，结果为 `0x00deadbe`
- 对第7至4位进行掩码处理，结果为 `0x00deadbe & 0x000000f0 = 0x000000b0`
- 对第7位进行符号扩展，结果为 `0xffffffffb0`

在软件中：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 第25至46行

```

25 void moving_mask() {
26 interp_config cfg = interp_default_config();
27 interp0->accum[0] = 0x1234abcd;
28
29 puts("掩码：");
30 printf("ACCUM0 = %08x\n", interp0->accum[0]);
31 for (int i = 0; i < 8; ++i) {
32 // 从最低有效位 (LSB) 到最高有效位 (MSB)。这些范围包含端点，因此0到31表示“整个32位寄存器”
33 interp_config_set_mask(&cfg, i * 4, i * 4 + 3);
34 interp_set_config(interp0, 0, &cfg);
35 // 从 ACCUMx_ADD 读取的是原始的通道偏移和掩码值，未加 BASEx
36
37 printf("Nibble %d: %08x\n", i, interp0->add_raw[0]);
38 }
39
39 puts("带符号扩展的掩码处理：");
40 interp_config_set_signed(&cfg, true);
41 for (int i = 0; i < 8; ++i) {
42 interp_config_set_mask(&cfg, i * 4, i * 4 + 3);
43 interp_set_config(interp0, 0, &cfg);
44 printf("Nibble %d: %08x\n", i, interp0->add_raw[0]);
45 }
46 }
```

上述示例应打印以下内容：

```

ACCUM0 = 1234abcd
Nibble 0: 0000000d
Nibble 1: 000000c0
Nibble 2: 00000b00
Nibble 3: 0000a000
Nibble 4: 00040000
Nibble 5: 00300000
Nibble 6: 02000000
Nibble 7: 10000000
带符号扩展的掩码:
Nibble 0: ffffffff 第 N 个
四位组 1: ffffffc0 第 2
个四位组: ffffffb00 第 3 个
四位组: fffffa000 第 4 个四
位组: 00040000 第 5 个四位
组: 00300000 第 6 个四位组
: 02000000 第 7 个四位组:
10000000

```

更改结果和输入多路复用器可以在累加器之间形成反馈。这一功能对于音频抖动处理非常有用。

Pico 示例: [https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp.c) 第 48 至 66 行

```

48 void cross_lanes() {
49 interp_config cfg = interp_default_config();
50 interp_config_set_cross_result(&cfg, true);
51 // ACCUM0 获取通道 1 的结果:
52 interp_set_config(interp0, 0, &cfg);
53 // ACCUM1 获取通道 0 的结果:
54 interp_set_config(interp0, 1, &cfg);
55
56 interp0->accum[0] = 123;
57 interp0->accum[1] = 456;
58 interp0->base[0] = 1;
59 interp0->base[1] = 0;
60 puts("车道结果交叉点: ");
61 for (int i = 0; i < 10; ++i) {
62 uint32_t peek0 = interp0->peek[0];
63 uint32_t pop1 = interp0->pop[1];
64 printf("PEEK0, POP1: %d, %d\n", peek0, pop1);
65 }
66 }

```

应打印以下内容:

```

PEEK0, POP1: 124, 456
PEEK0, POP1: 457, 124
PEEK0, POP1: 125, 457
PEEK0, POP1: 458, 125
PEEK0, POP1: 126, 458
PEEK0, POP1: 459, 126
PEEK0, POP1: 127, 459
PEEK0, POP1: 460, 127
PEEK0, POP1: 128, 460
PEEK0, POP1: 461, 128

```

### 3.1.10.2. 混合模式

混合模式可在每个核心的 `INTERP0` 上使用，并由 `CTRL_LANE0_BLEND` 控制标志启用。其执行线性插值，定义如下：

$$x = x_0 + \alpha(x_1 - x_0), \text{ for } 0 \leq \alpha < 1$$

其中为寄存器 `BASE0`，为寄存器 `BASE1`，且为由第1通道移位与掩码值的最低8位组成的分数值。

混合模式与普通模式的区别如下：

- `PEEK0, POP0` 返回8位alpha值（第1通道移位与掩码值的最低8位），结果的第31至24位为零。

- `PEEK1, POP1` 返回 `BASE0` 与 `BASE1` 之间的线性插值值。
- `PEEK2, POP2` 在相加时不包括第1通道的结果（即为 `BASE2 + 第0通道的移位和掩码值`）

当alpha值为0时，线性插值的结果等于 `BASE0`；当alpha值全为1时，结果等于 `BASE0 + 255/256 * (BASE1 - BASE0)`

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 68-87 行

```

68 void simple_blend1() {
69 puts("简单混合1:");
70
71 interp_config cfg = interp_default_config();
72 interp_config_set_blend(&cfg, true);
73 interp_set_config(interp0, 0, &cfg);
74
75 cfg = interp_default_config();
76 interp_set_config(interp0, 1, &cfg);
77
78 interp0->base[0] = 500;
79 interp0->base[1] = 1000;
80
81 for (int i = 0; i <= 6; i++) {
82 // 将 fraction 设置为介于 0 到 255 之间的值
83 interp0->accum[1] = 255 * i / 6;
84 // ≈ 500 + (1000 - 500) * i / 6;
85 printf("%d\n", (int) interp0->peek[1]);
86 }
87 }
```

应打印以下内容（注意 `255/256` 导致结果为 `998` 而非 `1000`）：

```

500
582
666
748
832
914
998
```

`CTRL_LANE1_SIGNED` 控制 `BASE0` 和 `BASE1` 是否对该插值结果进行符号扩展（此符号扩展是必需的，因为插值产生的中间乘积值为 40 位大小）。`CTRL_LANE0_SIGNED` 继续控制 `PEEK2` 和 `POP2` 中第 0 通道中间结果的符号扩展，功能正常。

Pico 示例: [https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 第 90 至 121 行

```

90 void print_simple_blend2_results(bool is_signed) {
91 //通道 1 的有符号标志控制基准 0/1 是否作为有符号或无符号处理
92 interp_config cfg = interp_default_config();
93 interp_config_set_signed(&cfg, is_signed);
94 interp_set_config(interp0, 1, &cfg);
95
96 for (int i = 0; i <= 6; i++) {
97 interp0->accum[1] = 255 * i / 6;
98 if (is_signed) {
99 printf("%d\n", (int) interp0->peek[1]);
100 } else {
101 printf("0x%08x\n", (uint) interp0->peek[1]);
102 }
103 }
104 }
105
106 void simple_blend2() {
107 puts("简单混合 2: ");
108
109 interp_config cfg = interp_default_config();
110 interp_config_set_blend(&cfg, true);
111 interp_set_config(interp0, 0, &cfg);
112
113 interp0->base[0] = (uint32_t) -1000;
114 interp0->base[1] = 1000;
115
116 puts("有符号: ");
117 print_simple_blend2_results(true);
118
119 puts("unsigned:");
120 print_simple_blend2_results(false);
121 }
```

应打印如下内容：

```

signed:
-1000
-672
-336
-8
328
656
992
unsigned:
0xfffffc18
0xd5ffffd60
0xaafffeb0
0x80fffff8
0x56000148
0x2c000290
0x010003e0
```

最后，在混合模式下，使用 **BASE\_1AND0** 寄存器通过单次32位写入向 **BASE0** 和 **BASE1** 分别发送16位值时，这些16位值在写入期间扩展为完整32位的符号扩展由**CTRL\_LANE1\_SIGNED**统一控制，区别于非混合模式操作中**CTRL\_LANE0\_SIGNED**控制对**BASE0**的扩展，及**CTRL\_LANE1\_SIGNED**控制对**BASE1**的扩展。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 行124-145

```

124 void simple_blend3() {
125 puts("简单混合3:");
126
127 interp_config cfg = interp_default_config();
128 interp_config_set_blend(&cfg, true);
129 interp_set_config(interp0, 0, &cfg);
130
131 cfg = interp_default_config();
132 interp_set_config(interp0, 1, &cfg);
133
134 interp0->accum[1] = 128;
135 interp0->base01 = 0x30005000;
136 printf("0x%08x\n", (int) interp0->peek[1]);
137 interp0->base01 = 0xe000f000;
138 printf("0x%08x\n", (int) interp0->peek[1]);
139
140 interp_config_set_signed(&cfg, true);
141 interp_set_config(interp0, 1, &cfg);
142
143 interp0->base01 = 0xe000f000;
144 printf("0x%08x\n", (int) interp0->peek[1]);
145 }
```

应打印如下内容：

```

0x00004000
0x0000e800
0xffffe800
```

### 3.1.10.3. 夹紧模式

夹紧模式适用于每个核上的 **INTERP1**。要启用夹紧模式，请将**CTRL\_LANE0\_CLAMP**控制标志设置为高电平。在钳位模式下，**PEEK0/POP0**的结果为通道值（经位移、掩码及符号扩展的**ACCUM0**），该值被限制在**BASE0**和**BASE1**之间。换言之，若通道值小于**BASE0**，则输出值为**BASE0**；若大于**BASE1**，则输出值为**BASE1**；否则，该值原样通过。不执行任何加法操作。这些比较的符号属性由**CTRL\_LANE0\_SIGNED**标志控制。

除此之外，插值器的行为与正常模式下保持一致。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 第193至211行

```

193 void clamp() {
194 puts("Clamp:");
195 interp_config cfg = interp_default_config();
196 interp_config_set_clamp(&cfg, true);
197 interp_config_set_shift(&cfg, 2);
198 // 根据符号位的新位置设置掩码。
199 interp_config_set_mask(&cfg, 0, 29);
200 // ...以确保移位后的值正确进行符号扩展。
201 interp_config_set_signed(&cfg, true);
202 interp_set_config(interp1, 0, &cfg);
203
204 interp1->base[0] = 0;
205 interp1->base[1] = 255;
206
207 for (int i = -1024; i <= 1024; i += 256) {
```

```

208 interp1->accum[0] = i;
209 printf("%d\t%d\n", i, (int) interp1->peek[0]);
210 }
211 }
```

应打印如下内容：

```

-1024 0
-768 0
-512 0
-256 0
0 0
256 64
512 128
768 192
1024 255
```

### 3.1.10.4. 示例用例：线性插值

线性插值结合了混合模式与其他插值器功能。在此示例中，`ACCUM0`跟踪要插值的值列表中的定点（整数/小数）位置。通道0用于生成位置整数部分的值数组地址。位置的小数部分被移位以产生用于混合的0至255之间的值。混合操作在数组中相邻的两个值之间进行。

最后，通过对`ACCUM0_ADD_RAW`的单次写操作更新小数位位置。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 第147至191行

```

147 void linear_interpolation() {
148 puts("线性插值: ");
149 const int uv_fractional_bits = 12;
150
151 // 用于通道0
152 // 移位并掩码 XXXX XXXX XXXX XXXX XXXX FFFF FFFF FFFF (累加0)
153 // 至 0000 0000 000X XXXX XXXX XXXX XXXX XXXX XXX0 154
// 即整数部分乘以2 (针对uint16_t) 155 interp_config cfg=inter
p_default_config();
156 interp_config_set_shift(&cfg, uv_fractional_bits - 1);
157 interp_config_set_mask(&cfg, 1, 32 - uv_fractional_bits);
158 interp_config_set_blend(&cfg, true);
159 interp_set_config(interp0, 0, &cfg);
160
161 // 针对通道1
162 // 将 XXXX XXXX XXXX XXXX XXXX FFFF FFFF FFFF (通过交叉输入累加为0) 左移
163 // 至 0000 XXXX XXXX XXXX XXXX FFFF FFFF FFFF
164
165 cfg = interp_default_config();
166 interp_config_set_shift(&cfg, uv_fractional_bits - 8);
167 interp_config_set_signed(&cfg, true);
168 interp_config_set_cross_input(&cfg, true); // 有符号混合
169 interp_set_config(interp0, 1, &cfg);
170
171 int16_t samples[] = {0, 10, -20, -1000, 500};
172
173 // step 是我们分数表示中的四分之一
174 uint step = (1 << uv_fractional_bits) / 4;
175
176 interp0->accum[0] = 0; // 初始 sample_offset;
```

```

177 interp0->base[2] = (uintptr_t) samples;
178 for (int i = 0; i < 16; i++) {
179 // result2 = samples + (lane0 原始结果)
180 // 即指向两个样本中第一个用于混合的指针
181 int16_t *sample_pair = (int16_t *) interp0->peek[2];
182 interp0->base[0] = sample_pair[0];
183 interp0->base[1] = sample_pair[1];
184 uint32_t peek1 = interp0->peek[1];
185 uint32_t add_raw1 = interp0->add_raw[1];
186 printf("%d\t(%d% 介于 %d 和 %d 之间)\n", (int) peek1,
187 100 * (add_raw1 & 0xff) / 0xff,
188 sample_pair[0], sample_pair[1]);
189 interp0->add_raw[0] = step;
190 }
191 }
```

应打印如下内容：

```

0 (0% 介于 0 和 10 之间)
2 (25% 介于 0 和 10 之间)
5 (50% 介于 0 和 10 之间)
7 (75% 介于 0 和 10 之间)
10 (0% 介于 10 和 -20 之间)
2 (25% 介于 10 和 -20 之间)
-5 (50% 介于 10 和 -20 之间)
-13 (75% 介于 10 和 -20 之间)
-20 (0% 介于 -20 和 -1000 之间)
-265 (25% 介于 -20 和 -1000 之间)
-510 (50% 介于 -20 和 -1000 之间)
-755 (在-20和-1000之间的75%)
-1000 (在-1000和500之间的0%)
-625 (在-1000和500之间的25%)
-250 (在-1000和500之间的50%)
125 (在-1000和500之间的75%)
```

此方法用于SDK中快速近似的音频上采样。

### 3.1.10.5. 示例用例：简单仿射纹理映射

简单仿射纹理映射可通过对纹理坐标进行定点运算，并在扫描线的每个像素上按固定量递增坐标来实现。纹理坐标的整数部分构成纹理地址。从 `POP2` 读取时，会将偏移量加至纹理基指针。处理器加载所得地址以从纹理中采样像素颜色。

通过使用两个通道、所有三个基值以及 `CTRL_LANEx_ADD_RAW` 标志，可利用插值器将高耗时的CPU操作简化为单周期迭代。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\\_interp/hello\\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c) 第214至272行

```

214 void texture_mapping_setup(uint8_t *texture, uint texture_width_bits, uint
215 texture_height_bits,
216 uint uv_fractional_bits) {
217 interp_config cfg = interp_default_config();
218 // 设置 add_raw 标志，以使用原始（未移位且未掩码）通道累加器值进行加法
219 interp_config_set_add_raw(&cfg, true);
220 interp_config_set_shift(&cfg, uv_fractional_bits);
```

```

221 interp_config_set_mask(&cfg, 0, texture_width_bits - 1);
222 interp_set_config(interp0, 0, &cfg);
223
224 interp_config_set_shift(&cfg, uv_fractional_bits - texture_width_bits);
225 interp_config_set_mask(&cfg, texture_width_bits, texture_width_bits +
226 texture_height_bits - 1);
227 interp_set_config(interp0, 1, &cfg);
228
229 interp0->base[2] = (uintptr_t) texture;
230 }
231
232 void texture_mapped_span(uint8_t *output, uint32_t u, uint32_t v, uint32_t du, uint32_t dv,
233 uint count) {
234 // u, v 是具有 uv_fractional_bits 小数位的定点纹理坐标。
235 // du, dv 是沿跨度方向的纹理坐标步长，采用相同的定点格式。
236 interp0->accum[0] = u;
237 interp0->base[0] = du;
238 interp0->accum[1] = v;
239 interp0->base[1] = dv;
240 for (uint i = 0; i < count; i++) {
241 // 等效于
242 // uint32_t sm_result0 = (accum0 >> uv_fractional_bits) & (1 << (texture_width_bits -
243 1));
244 // uint32_t sm_result1 = (accum1 >> uv_fractional_bits) & (1 << (texture_height_bits -
245 1));
246 // uint8_t *address = texture + sm_result0 + (sm_result1 << texture_width_bits);
247 // output[i] = *address;
248 // accum0 = du + accum0;
249 // accum1 = dv + accum1;
250
251 }
252
253 void texture_mapping() {
254 puts("仿射纹理映射（含纹理重复）：");
255
256 uint8_t texture[] = {
257 0x00, 0x01, 0x02, 0x03,
258 0x10, 0x11, 0x12, 0x13,
259 0x20, 0x21, 0x22, 0x23,
260 0x30, 0x31, 0x32, 0x33,
261 };
262 // 4x4 纹理
263 texture_mapping_setup(texture, 2, 2, 16);
264 uint8_t output[12];
265 uint32_t du = 65536 / 2; // 1/2 步长
266 uint32_t dv = 65536 / 3; // 1/3 步长
267 texture_mapped_span(output, 0, 0, du, dv, 12);
268
269 for (uint i = 0; i < 12; i++) {
270 printf("0x%02x\n", output[i]);
271 }
272 }

```

应打印如下内容：

```

0x00
0x00
0x01
0x01
0x12
0x12
0x12
0x13
0x23
0x20
0x20
0x31
0x31

```

### 3.1.11. 寄存器列表

SIO 寄存器起始于基地址 `0xd0000000`（在 SDK 中定义为 SIO\_BASE）。

表 17. SIO  
寄存器清单

| 偏移量   | 名称                              | 说明                                                                                                                                                                                                                                                                    |
|-------|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x000 | <a href="#">CPUID</a>           | 处理器内核标识符                                                                                                                                                                                                                                                              |
| 0x004 | <a href="#">GPIO_IN</a>         | GPIO0...31 的输入值。<br><br>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 显示为零。                                                                                                                                                                                                     |
| 0x008 | <a href="#">GPIO_HI_IN</a>      | GPIO32...47、QSPI IO 及 USB 引脚的输入值<br><br>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 显示为零。                                                                                                                                                                                     |
| 0x010 | <a href="#">GPIO_OUT</a>        | GPIO0...31 的输出值                                                                                                                                                                                                                                                       |
| 0x014 | <a href="#">GPIO_HI_OUT</a>     | GPIO32...47、QSPI IO 及 USB 引脚的输出值。<br><br>写入以设置输出电平（1/0 → 高/低）。读取返回的是最后写入的值，而非引脚的输入值。如果内核 0 和内核 1 同时写入 GPIO_HI_OUT（或 SET/CLR/XOR 别名），结果相当于先执行内核 0 的写入，然后再将内核 1 的写入应用于该中间结果。<br><br>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 会忽略写操作，其输出状态读取返回零。此条款同样适用于该寄存器的 SET/CLR/XOR 别名。 |
| 0x018 | <a href="#">GPIO_OUT_SET</a>    | GPIO0...31 输出值设置                                                                                                                                                                                                                                                      |
| 0x01c | <a href="#">GPIO_HI_OUT_SET</a> | GPIO32..47、QSPI IO 及 USB 引脚的输出值设置。<br>对 GPIO_HI_OUT 执行原子位设置操作，即 <code>GPIO_HI_OUT  = wdata</code>                                                                                                                                                                     |
| 0x020 | <a href="#">GPIO_OUT_CLR</a>    | GPIO0...31 输出值清除                                                                                                                                                                                                                                                      |
| 0x024 | <a href="#">GPIO_HI_OUT_CLR</a> | GPIO32..47、QSPI IO 及 USB 引脚的输出值清除。<br>对 GPIO_HI_OUT 执行原子位清除操作，即 <code>GPIO_HI_OUT &amp;= ~wdata</code>                                                                                                                                                                |
| 0x028 | <a href="#">GPIO_OUT_XOR</a>    | GPIO0...31 输出值异或                                                                                                                                                                                                                                                      |

| 偏移量   | 名称                 | 说明                                                                                                                                                                                                                                                    |
|-------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x02c | GPIO_HI_OUT_XOR    | GPIO32..47、QSPI IO 及 USB 引脚的输出值异或。<br>对 GPIO_HI_OUT 执行原子位异或操作，即 <code>GPIO_HI_OUT ^= wdata</code>                                                                                                                                                     |
| 0x030 | GPIO_OE            | GPIO0...31 输出使能                                                                                                                                                                                                                                       |
| 0x034 | GPIO_HI_OE         | GPIO32...47、QSPI IO 和 USB 引脚的输出使能值。<br><br>写入输出使能（1/0 → 输出/输入）。读回值为最后写入的数值。若核心0与核心1同时写入GPIO_HI_OE（或对应的SET/CLR/XOR别名寄存器），结果相当于先执行核心0的写入，随后在该中间结果基础上执行核心1的写入。<br><br>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 会忽略写操作，其输出状态读取返回零。此条款同样适用于该寄存器的SET/CLR/XOR 别名。 |
| 0x038 | GPIO_OE_SET        | GPIO0...31 输出使能设置                                                                                                                                                                                                                                     |
| 0x03c | GPIO_HI_OE_SET     | GPIO32...47、QSPI IO 和 USB 引脚的输出使能设置。<br>对 GPIO_HI_OE 执行原子位设置，即 <code>GPIO_HI_OE  = wdata</code>                                                                                                                                                       |
| 0x040 | GPIO_OE_CLR        | GPIO0...31 输出使能清除                                                                                                                                                                                                                                     |
| 0x044 | GPIO_HI_OE_CLR     | GPIO32...47、QSPI IO 和 USB 引脚的输出使能清除。<br>对 GPIO_HI_OE 执行原子位清除，即 <code>GPIO_HI_OE &amp;= ~wdata</code>                                                                                                                                                  |
| 0x048 | GPIO_OE_XOR        | GPIO0...31 输出使能异或                                                                                                                                                                                                                                     |
| 0x04c | GPIO_HI_OE_XOR     | GPIO32...47、QSPI IO 和 USB 引脚的输出使能异或<br>对 GPIO_HI_OE 执行原子位异或操作，即 <code>GPIO_HI_OE ^= wdata</code>                                                                                                                                                      |
| 0x050 | FIFO_ST            | 核间 FIFO（邮箱）的状态寄存器                                                                                                                                                                                                                                     |
| 0x054 | FIFO_WR            | 对本核 TX FIFO 的写访问                                                                                                                                                                                                                                      |
| 0x058 | FIFO_RD            | 对本核 RX FIFO 的读访问                                                                                                                                                                                                                                      |
| 0x05c | SPINLOCK_ST        | 自旋锁状态                                                                                                                                                                                                                                                 |
| 0x080 | INTERP0_ACCUM0     | 累加器 0 的读写访问                                                                                                                                                                                                                                           |
| 0x084 | INTERP0_ACCUM1     | 累加器 1 的读写访问                                                                                                                                                                                                                                           |
| 0x088 | INTERP0_BASE0      | 对 BASE0 寄存器的读写访问。                                                                                                                                                                                                                                     |
| 0x08c | INTERP0_BASE1      | 对 BASE1 寄存器的读写访问。                                                                                                                                                                                                                                     |
| 0x090 | INTERP0_BASE2      | 对 BASE2 寄存器的读写访问。                                                                                                                                                                                                                                     |
| 0x094 | INTERP0_POP_LANE0  | 读取 LANE0 结果，同时将该通道结果写入两个累加器（POP 操作）。                                                                                                                                                                                                                  |
| 0x098 | INTERP0_POP_LANE1  | 读取 LANE1 结果，同时将该通道结果写入两个累加器（POP 操作）。                                                                                                                                                                                                                  |
| 0x09c | INTERP0_POP_FULL   | 读取 FULL 结果，同时将该通道结果写入两个累加器（POP 操作）。                                                                                                                                                                                                                   |
| 0x0a0 | INTERP0_PEEK_LANE0 | 读取 LANE0 结果，不改变任何内部状态（PEEK 操作）。                                                                                                                                                                                                                       |
| 0x0a4 | INTERP0_PEEK_LANE1 | 读取 LANE1 结果，不改变任何内部状态（PEEK 操作）。                                                                                                                                                                                                                       |

| 偏移量   | 名称                 | 说明                                |
|-------|--------------------|-----------------------------------|
| 0x0a8 | INTERP0_PEEK_FULL  | 读取FULL结果，不改变任何内部状态（PEEK操作）。       |
| 0x0ac | INTERP0_CTRL_LANE0 | 通道0控制寄存器                          |
| 0x0b0 | INTERP0_CTRL_LANE1 | 通道1控制寄存器                          |
| 0x0b4 | INTERP0_ACCUM0_ADD | 写入的值将以原子方式累加至ACCUM0               |
| 0x0b8 | INTERP0_ACCUM1_ADD | 写入的值将以原子方式累加至ACCUM1               |
| 0x0bc | INTERP0_BASE_1AND0 | 写入时，低16位同时写入BASE0，高位写入BASE1。      |
| 0x0c0 | INTERP1_ACCUM0     | 累加器0的读写访问                         |
| 0x0c4 | INTERP1_ACCUM1     | 累加器1的读写访问                         |
| 0x0c8 | INTERP1_BASE0      | 对BASE0寄存器的读写访问。                   |
| 0x0cc | INTERP1_BASE1      | 对BASE1寄存器的读写访问。                   |
| 0x0d0 | INTERP1_BASE2      | 对BASE2寄存器的读写访问。                   |
| 0x0d4 | INTERP1_POP_LANE0  | 读取LANE0结果，同时将该通道结果写入两个累加器（POP操作）。 |
| 0x0d8 | INTERP1_POP_LANE1  | 读取LANE1结果，同时将该通道结果写入两个累加器（POP操作）。 |
| 0x0dc | INTERP1_POP_FULL   | 读取FULL结果，同时将该通道结果写入两个累加器（POP操作）。  |
| 0x0e0 | INTERP1_PEEK_LANE0 | 读取LANE0结果，不改变任何内部状态（PEEK操作）。      |
| 0x0e4 | INTERP1_PEEK_LANE1 | 读取LANE1结果，不改变任何内部状态（PEEK操作）。      |
| 0x0e8 | INTERP1_PEEK_FULL  | 读取FULL结果，不改变任何内部状态（PEEK操作）。       |
| 0x0ec | INTERP1_CTRL_LANE0 | 通道0控制寄存器                          |
| 0x0f0 | INTERP1_CTRL_LANE1 | 通道1控制寄存器                          |
| 0x0f4 | INTERP1_ACCUM0_ADD | 写入的值将以原子方式累加至ACCUM0               |
| 0x0f8 | INTERP1_ACCUM1_ADD | 写入的值将以原子方式累加至ACCUM1               |
| 0x0fc | INTERP1_BASE_1AND0 | 写入时，低16位同时写入BASE0，高位写入BASE1。      |
| 0x100 | SPINLOCK0          | 自旋锁寄存器0                           |
| 0x104 | SPINLOCK1          | 自旋锁寄存器1                           |
| 0x108 | SPINLOCK2          | 自旋锁寄存器2                           |
| 0x10c | SPINLOCK3          | 自旋锁寄存器3                           |
| 0x110 | SPINLOCK4          | 自旋锁寄存器4                           |
| 0x114 | SPINLOCK5          | 自旋锁寄存器5                           |
| 0x118 | SPINLOCK6          | 自旋锁寄存器6                           |
| 0x11c | SPINLOCK7          | 自旋锁寄存器7                           |
| 0x120 | SPINLOCK8          | 自旋锁寄存器8                           |
| 0x124 | SPINLOCK9          | 自旋锁寄存器9                           |

| 偏移量   | 名称               | 说明                                                                                                                                         |
|-------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 0x128 | SPINLOCK10       | 自旋锁寄存器10                                                                                                                                   |
| 0x12c | SPINLOCK11       | 自旋锁寄存器11                                                                                                                                   |
| 0x130 | SPINLOCK12       | 自旋锁寄存器12                                                                                                                                   |
| 0x134 | SPINLOCK13       | 自旋锁寄存器13                                                                                                                                   |
| 0x138 | SPINLOCK14       | 自旋锁寄存器14                                                                                                                                   |
| 0x13c | SPINLOCK15       | 自旋锁寄存器 15                                                                                                                                  |
| 0x140 | SPINLOCK16       | 自旋锁寄存器 16                                                                                                                                  |
| 0x144 | SPINLOCK17       | 自旋锁寄存器 17                                                                                                                                  |
| 0x148 | SPINLOCK18       | 自旋锁寄存器 18                                                                                                                                  |
| 0x14c | SPINLOCK19       | 自旋锁寄存器 19                                                                                                                                  |
| 0x150 | SPINLOCK20       | 自旋锁寄存器 20                                                                                                                                  |
| 0x154 | SPINLOCK21       | 自旋锁寄存器 21                                                                                                                                  |
| 0x158 | SPINLOCK22       | 自旋锁寄存器 22                                                                                                                                  |
| 0x15c | SPINLOCK23       | 自旋锁寄存器 23                                                                                                                                  |
| 0x160 | SPINLOCK24       | 自旋锁寄存器 24                                                                                                                                  |
| 0x164 | SPINLOCK25       | 自旋锁寄存器 25                                                                                                                                  |
| 0x168 | SPINLOCK26       | 自旋锁寄存器 26                                                                                                                                  |
| 0x16c | SPINLOCK27       | 自旋锁寄存器27                                                                                                                                   |
| 0x170 | SPINLOCK28       | 自旋锁寄存器28                                                                                                                                   |
| 0x174 | SPINLOCK29       | 自旋锁寄存器29                                                                                                                                   |
| 0x178 | SPINLOCK30       | 自旋锁寄存器30                                                                                                                                   |
| 0x17c | SPINLOCK31       | 自旋锁寄存器31                                                                                                                                   |
| 0x180 | DOORBELL_OUT_SET | <p>触发对侧核心的门铃中断。</p> <p>向某位写入1以设置对侧核心 DOORBELL_IN 寄存器中对应的位。此操作将引发对侧核心的门铃中断。</p> <p>读取该寄存器以获取对侧核心当前激活的门铃状态。这相当于该核心读取其自身的 DOORBELL_IN 状态。</p> |

| 偏移量   | 名称               | 说明                                                                                                                                                                                                                                                                     |
|-------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x184 | DOORBELL_OUT_CLR | <p>清除已发送至对侧核心的门铃信号。<br/>该寄存器专用于调试和初始化。</p> <p>向 DOORBELL_OUT_CLR 中的某位写入 1，清除对侧核心 DOORBELL_IN 中对应的位。<br/>清除所有位将导致该核心的门铃中断撤销。鉴于软件通常先通过 DOORBELL_OUT_SET 发送事件，再通过向 DOORBELL_IN_CLR 写入进行事件确认，因此应谨慎使用此寄存器，以避免竞态条件。</p> <p>读取返回另一核心当前已触发的门铃状态，即该核心读取其自身的 DOORBELL_IN 状态。</p> |
| 0x188 | DOORBELL_IN_SET  | 写入 1 以触发本核心的门铃中断。读取以获取本核心当前已触发的门铃状态。                                                                                                                                                                                                                                   |
| 0x18c | DOORBELL_IN_CLR  | <p>检查并确认发送至该核心的门铃信号。当本寄存器中任一位为1时，该核心的门铃中断即被触发。</p> <p>写入1至每个位以清除此位。全部位清除后，门铃中断即取消触发。读取以获取本核心当前已触发的门铃状态。</p>                                                                                                                                                            |
| 0x190 | PERI_NONSEC      | <p>将某些核心本地外设从安全SIO断开，并连接至非安全SIO，以使非安全软件得以使用。当外设连接到非安全SIO时，若从安全SIO访问该外设，反之亦然，均会导致总线错误。</p> <p>此寄存器为每核心专属，且仅存在于安全SIO内。</p> <p>大多数SIO硬件在安全与非安全SIO中均有复制，故未在本寄存器中列出。</p>                                                                                                    |
| 0x1a0 | RISCV_SOFTIRQ    | <p>控制RISC-V核心上标准软件中断（MIP.MSIP）的触发与否。</p> <p>与RISC-V定时器不同，此中断未被路由至标准系统级中断线，因此Arm核心无法使用。</p> <p>两个核心在同一周期内写入该寄存器是安全的。置位/清除效果在两个核心之间累积后应用。若某标志在同一周期既被置位又被清除，则仅置位生效。</p>                                                                                                   |

| 偏移量   | 名称                               | 说明                                                                                                                                                      |
|-------|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x1a4 | <a href="#">MTIME_CTRL</a>       | RISC-V 64位机器模式计时器控制寄存器。该计时器仅存在于安全SIO中，因此仅可由处于安全模式的Arm核心或处于机器模式的RISC-V核心访问。<br><br>注意，尽管该计时器遵循RISC-V特权规范，但同样适用于Arm核心。中断被路由至普通系统级中断线及RISC-V核心的MIP.MTIP输入。 |
| 0x1b0 | <a href="#">MTIME</a>            | 对RISC-V机器模式计时器高半部的读写访问权限。该寄存器由两个核心共享。若两个核心在同一周期内写入，则核心1优先。                                                                                              |
| 0x1b4 | <a href="#">MTIMEH</a>           | 对RISC-V机器模式计时器高半部的读写访问权限。该寄存器由两个核心共享。若两个核心在同一周期内写入，则核心1优先。                                                                                              |
| 0x1b8 | <a href="#">MTIMECMP</a>         | RISC-V 机器模式定时器比较器的低半部分。该寄存器为核心本地寄存器，即每个核心均拥有该寄存器的副本，比较结果会传递至对应核心的中断线。<br><br>当 MTIME 大于或等于 MTIMECMP 时，将产生定时器中断。该比较为无符号比较，基于完整的 64 位值执行。                 |
| 0x1bc | <a href="#">MTIMECMPH</a>        | RISC-V 机器模式定时器比较器的高半部分。该寄存器为核心本地寄存器。<br><br>当 MTIME 大于或等于 MTIMECMP 时，将产生定时器中断。该比较为无符号比较，基于完整的 64 位值执行。                                                  |
| 0x1c0 | <a href="#">TMDS_CTRL</a>        | TMDS 编码器的控制寄存器。                                                                                                                                         |
| 0x1c4 | <a href="#">TMDS_WDATA</a>       | 对 TMDS 颜色数据寄存器的只写访问。                                                                                                                                    |
| 0x1c8 | <a href="#">TMDS_PEEK_SINGLE</a> | 获取一个像素的颜色数据编码，封装为 32 位值（3 个 10 位符号）。<br><br>PEEK 别名在读取时不会移动颜色寄存器，但仍会更新每个编码器的直流平衡状态。<br>此功能对像素倍增非常有用。                                                    |
| 0x1cc | <a href="#">TMDS_POP_SINGLE</a>  | 获取一个像素的颜色数据编码，打包成32位数值。打包格式为5个块，每块包含3条通道，每条通道2位（共计30位）。每个块包含每条通道对应TMDS符号的两位。该格式设计用于RP2350上的HSTX外设进行移位输出。<br><br>POP别名在读取时会移位颜色寄存器，并推进每个编码器的直流平衡状态。      |

| 偏移量   | 名称                  | 说明                                                                                                                                                   |
|-------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x1d0 | TMDS_PEEK_DOUBLE_L0 | <p>获取两像素颜色数据编码中的第0通道。<br/>两个10位的TMDS符号被打包在32位字的低位。</p> <p>PEEK别名读取时不移位颜色寄存器，但仍然会推进第0通道的直<br/>流平衡状态。当需要一次性读取所有3条通道的编码，而非逐条<br/>通道处理整行扫描时，此功能尤为有用。</p> |
| 0x1d4 | TMDS_POP_DOUBLE_L0  | <p>获取两像素颜色数据编码中的第0通道。<br/>两个10位的TMDS符号被打包在32位字的低位。</p> <p>POP别名在读取时根据PIX_SHIFT和PIX2_NOSHIFT的值对颜色<br/>寄存器进行偏移。</p>                                    |
| 0x1d8 | TMDS_PEEK_DOUBLE_L1 | <p>获取两个像素颜色数据编码的第1通道。<br/>两个10位的TMDS符号被打包在32位字的低位。</p> <p>PEEK别名在读取时不偏移颜色寄存器，但仍会推进第1通道的D<br/>C平衡状态。当需要一次性读取所有3条通道的编码，而非逐条<br/>通道处理整行扫描时，此功能尤为有用。</p> |
| 0x1dc | TMDS_POP_DOUBLE_L1  | <p>获取两个像素颜色数据编码的第1通道。<br/>两个10位的TMDS符号被打包在32位字的低位。</p> <p>POP别名在读取时根据PIX_SHIFT和PIX2_NOSHIFT的值对颜色<br/>寄存器进行偏移。</p>                                    |
| 0x1e0 | TMDS_PEEK_DOUBLE_L2 | <p>获取两个像素颜色数据编码的第2通道。<br/>两个10位的TMDS符号被打包在32位字的低位。</p> <p>PEEK别名在读取时不偏移颜色寄存器，但仍会推进第2通道的D<br/>C平衡状态。当需要一次性读取所有3条通道的编码，而非逐条<br/>通道处理整行扫描时，此功能尤为有用。</p> |
| 0x1e4 | TMDS_POP_DOUBLE_L2  | <p>获取两个像素颜色数据编码的第2通道。<br/>两个10位的TMDS符号被打包在32位字的低位。</p> <p>POP别名在读取时根据PIX_SHIFT和PIX2_NOSHIFT的值对颜色<br/>寄存器进行偏移。</p>                                    |

## SIO: CPUID寄存器

偏移: 0x000

### 描述

处理器核心标识

表18. CPUID  
寄存器

| 位    | 描述                            | 类型 | 复位 |
|------|-------------------------------|----|----|
| 31:0 | 从处理器核心0读取时值为0， 从处理器核心1读取时值为1。 | 只读 | -  |

## SIO: GPIO\_IN 寄存器

偏移量: 0x004

表19. GPIO\_IN  
寄存器

| 位    | 描述                                                                | 类型 | 复位         |
|------|-------------------------------------------------------------------|----|------------|
| 31:0 | GPIO0...31 的输入值。<br><br>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 显示为零。 | 只读 | 0x00000000 |

## SIO: GPIO\_HI\_IN 寄存器

偏移量: 0x008

### 说明

GPIO32...47、QSPI IO 及 USB 引脚的输入值

在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 显示为零。

表20. GPIO\_HI\_IN  
寄存器

| 位     | 描述                                                           | 类型 | 复位     |
|-------|--------------------------------------------------------------|----|--------|
| 31:28 | <b>QSPI_SD</b> : QSPI SD0 (MOSI)、SD1 (MISO)、SD2 和 SD3 引脚的输入值 | 只读 | 0x0    |
| 27    | <b>QSPI_CSN</b> : QSPI CSn 引脚的输入值                            | 只读 | 0x0    |
| 26    | <b>QSPI_SCK</b> : QSPI SCK 引脚的输入值                            | 只读 | 0x0    |
| 25    | <b>USB_DM</b> : USB D- 引脚的输入值                                | 只读 | 0x0    |
| 24    | <b>USB_DP</b> : USB D+ 引脚的输入值                                | 只读 | 0x0    |
| 23:16 | 保留。                                                          | -  | -      |
| 15:0  | <b>GPIO</b> : GPIO32至47 引脚的输入值                               | 只读 | 0x0000 |

## SIO: GPIO\_OUT 寄存器

偏移量: 0x010

### 描述

GPIO0...31 的输出值

表21. GPIO\_OUT 寄存器

| 位    | 描述                                                                                                                                                                                                                                        | 类型 | 复位         |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | <p>设置GPIO0至31的输出电平（1/0 → 高/低）。读取时返回最后写入的值，而非引脚的输入值。</p> <p>若核心0与核心1同时写入GPIO_OUT寄存器（或对应的SET/CLR/XO R别名寄存器），结果等同于核心0的写入先执行，随后核心1的写入应用于该中间结果。</p> <p>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 会忽略写操作，其输出状态读取返回零。此条款同样适用于该寄存器的 SET/CLR/XOR 别名。</p> | 读写 | 0x00000000 |

## SIO: GPIO\_HI\_OUT 寄存器

偏移量：0x014

### 描述

GPIO32...47、QSPI IO 及 USB 引脚的输出值。

写入以设置输出电平（1/0 → 高/低）。读取时返回最后写入的值，而非引脚的输入值。

如果内核 0 和内核 1 同时写入 GPIO\_HI\_OUT（或 SET/CLR/XOR 别名），结果相当于先执行内核 0 的写入，随后再将内核 1 的写入应用于该中间结果。

在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 会忽略写操作，其输出状态读取返回零。此条款同样适用于该寄存器的 SET/CLR/XOR 别名。

表22. GPIO\_HI\_OUT 寄存器

| 位     | 描述                                                           | 类型 | 复位     |
|-------|--------------------------------------------------------------|----|--------|
| 31:28 | <b>QSPI_SD</b> : QSPI SD0 (MOSI)、SD1 (MISO)、SD2 及 SD3 引脚的输出值 | 读写 | 0x0    |
| 27    | <b>QSPI_CSN</b> : QSPI CSn 引脚的输出值                            | 读写 | 0x0    |
| 26    | <b>QSPI_SCK</b> : QSPI SCK 引脚的输出值                            | 读写 | 0x0    |
| 25    | <b>USB_DM</b> : USB D- 引脚的输出值                                | 读写 | 0x0    |
| 24    | <b>USB_DP</b> : USB D+ 引脚的输出值                                | 读写 | 0x0    |
| 23:16 | 保留。                                                          | -  | -      |
| 15:0  | <b>GPIO</b> : GPIO32 至 47 的输出值                               | 读写 | 0x0000 |

## SIO: GPIO\_OUT\_SET 寄存器

偏移量：0x018

### 描述

GPIO0...31 输出值设置

表23. GPIO\_OUT\_SET 寄存器

| 位    | 描述                                                    | 类型 | 复位         |
|------|-------------------------------------------------------|----|------------|
| 31:0 | 对 GPIO_OUT 执行原子位设置操作，即 <code>GPIO_OUT  = wdata</code> | WO | 0x00000000 |

## SIO: GPIO\_HI\_OUT\_SET 寄存器

偏移量：0x01C

### 描述

GPIO32..47、QSPI IO 及 USB 引脚的输出值设置。

对 GPIO\_HI\_OUT 执行原子位设置操作，即 `GPIO_HI_OUT |= wdata`

表 24  
GPIO\_HI\_OUT\_SET  
寄存器

| 位     | 描述              | 类型 | 复位     |
|-------|-----------------|----|--------|
| 31:28 | <b>QSPI_SD</b>  | WO | 0x0    |
| 27    | <b>QSPI_CSN</b> | WO | 0x0    |
| 26    | <b>QSPI_SCK</b> | WO | 0x0    |
| 25    | <b>USB_DM</b>   | WO | 0x0    |
| 24    | <b>USB_DP</b>   | WO | 0x0    |
| 23:16 | 保留。             | -  | -      |
| 15:0  | <b>GPIO</b>     | WO | 0x0000 |

## SIO: GPIO\_OUT\_CLR 寄存器

偏移量: 0x020

### 描述

GPIO0...31 输出值清除

表 25  
GPIO\_OUT\_CLR  
寄存器

| 位    | 描述                                                         | 类型 | 复位         |
|------|------------------------------------------------------------|----|------------|
| 31:0 | 对 GPIO_OUT 执行原子位清除操作，即 <code>GPIO_OUT &amp;= ~wdata</code> | WO | 0x00000000 |

## SIO: GPIO\_HI\_OUT\_CLR 寄存器

偏移量: 0x024

### 说明

GPIO32..47、QSPI IO 及 USB 引脚的输出值清除。

对 GPIO\_HI\_OUT 执行原子位清除操作，即 `GPIO_HI_OUT &= ~wdata`

表 26。  
GPIO\_HI\_OUT\_CLR  
寄存器

| 位     | 描述              | 类型 | 复位     |
|-------|-----------------|----|--------|
| 31:28 | <b>QSPI_SD</b>  | WO | 0x0    |
| 27    | <b>QSPI_CSN</b> | WO | 0x0    |
| 26    | <b>QSPI_SCK</b> | WO | 0x0    |
| 25    | <b>USB_DM</b>   | WO | 0x0    |
| 24    | <b>USB_DP</b>   | WO | 0x0    |
| 23:16 | 保留。             | -  | -      |
| 15:0  | <b>GPIO</b>     | WO | 0x0000 |

## SIO: GPIO\_OUT\_XOR 寄存器

偏移量: 0x028

### 说明

GPIO0...31 输出值异或

表 27。  
GPIO\_OUT\_XOR  
寄存器

| 位    | 描述                                                    | 类型 | 复位         |
|------|-------------------------------------------------------|----|------------|
| 31:0 | 对 GPIO_OUT 执行原子位异或操作，即 <code>GPIO_OUT ^= wdata</code> | WO | 0x00000000 |

## SIO: GPIO\_HI\_OUT\_XOR 寄存器

偏移量: 0x02c

**描述**

GPIO32..47、QSPI IO 及 USB 引脚的输出值异或。  
对 GPIO\_HI\_OUT 执行原子位异或操作，即 `GPIO_HI_OUT ^= wdata`

表 28。  
*GPIO\_HI\_OUT\_XOR*  
寄存器

| 位     | 描述               | 类型 | 复位     |
|-------|------------------|----|--------|
| 31:28 | <b>QSPI_SD</b>   | WO | 0x0    |
| 27    | <b>QSPI_CS_N</b> | WO | 0x0    |
| 26    | <b>QSPI_SCK</b>  | WO | 0x0    |
| 25    | <b>USB_DM</b>    | WO | 0x0    |
| 24    | <b>USB_DP</b>    | WO | 0x0    |
| 23:16 | 保留。              | -  | -      |
| 15:0  | <b>GPIO</b>      | WO | 0x0000 |

**SIO: GPIO\_OE 寄存器**

偏移量：0x030

**描述**

GPIO0...31 输出使能

表 29。*GPIO\_OE*  
寄存器

| 位    | 描述                                                                                                                                                                                                                                     | 类型 | 复位         |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | <p>设置 GPIO0...31 的输出使能（1/0 → 输出/输入）；读取时返回最后写入的值。</p> <p>若核 0 和核 1 同时写入 GPIO_OE（或写入其 SET/CLR/XOR 别名），结果相当于先执行核 0 的写入，然后将核 1 的写入应用于该中间结果。</p> <p>在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 会忽略写操作，其输出状态读取返回零。此条款同样适用于该寄存器的 SET/CLR/XOR 别名。</p> | 读写 | 0x00000000 |

**SIO: GPIO\_HI\_OE 寄存器**

偏移：0x034

**说明**

GPIO32...47、QSPI IO 和 USB 引脚的输出使能值。

写入输出使能（1/0 → 输出/输入）。读回值为最后写入的数值。若核心0与核心1同时写入GPIO\_HI\_OE（或对应的SET/CLR/XOR别名寄存器），结果相当于先执行核心0的写入，随后在该中间结果基础上执行核心1的写入。

在非安全 SIO 中，依据 ACCESSCTRL 的安全专用 GPIO 会忽略写操作，其输出状态读取返回零。此条款同样适用于该寄存器的SET/CLR/XOR别名。

表 30。*GPIO\_HI\_OE*  
寄存器

| 位     | 描述                                                          | 类型 | 复位  |
|-------|-------------------------------------------------------------|----|-----|
| 31:28 | <b>QSPI_SD</b> : QSPI SD0 (MOSI)、SD1 (MISO)、SD2及SD3引脚的输出使能值 | 读写 | 0x0 |
| 27    | <b>QSPI_CS_N</b> : QSPI CSn引脚的输出使能值                         | 读写 | 0x0 |
| 26    | <b>QSPI_SCK</b> : QSPI SCK引脚的输出使能值                          | 读写 | 0x0 |
| 25    | <b>USB_DM</b> : USB D-引脚的输出使能值                              | 读写 | 0x0 |

| 位     | 描述                             | 类型 | 复位     |
|-------|--------------------------------|----|--------|
| 24    | <b>USB_DP</b> : USB D+引脚的输出使能值 | 读写 | 0x0    |
| 23:16 | 保留。                            | -  | -      |
| 15:0  | <b>GPIO</b> : GPIO32至47的输出使能值  | 读写 | 0x0000 |

## SIO: GPIO\_OE\_SET 寄存器

偏移: 0x038

### 说明

GPIO0...31 输出使能设置

表31.  
GPIO\_OE\_SET 寄存器

| 位    | 描述                                                      | 类型 | 复位         |
|------|---------------------------------------------------------|----|------------|
| 31:0 | 对 GPIO_OE 执行原子位设置操作，即 $\text{GPIO\_OE}  = \text{wdata}$ | WO | 0x00000000 |

## SIO: GPIO\_HI\_OE\_SET 寄存器

偏移: 0x03c

### 说明

GPIO32...47、QSPI IO 和 USB 引脚的输出使能设置。

对 GPIO\_HI\_OE 执行原子位设置，即  $\text{GPIO\_HI\_OE} |= \text{wdata}$

表32.  
GPIO\_HI\_OE\_SET 寄存器

| 位     | 描述              | 类型 | 复位     |
|-------|-----------------|----|--------|
| 31:28 | <b>QSPI_SD</b>  | WO | 0x0    |
| 27    | <b>QSPI_CSN</b> | WO | 0x0    |
| 26    | <b>QSPI_SCK</b> | WO | 0x0    |
| 25    | <b>USB_DM</b>   | WO | 0x0    |
| 24    | <b>USB_DP</b>   | WO | 0x0    |
| 23:16 | 保留。             | -  | -      |
| 15:0  | <b>GPIO</b>     | WO | 0x0000 |

## SIO: GPIO\_OE\_CLR 寄存器

偏移量: 0x040

### 说明

GPIO0...31 输出使能清除

表 33.  
GPIO\_OE\_CLR 寄存器

| 位    | 描述                                                           | 类型 | 复位         |
|------|--------------------------------------------------------------|----|------------|
| 31:0 | 对 GPIO_OE 执行原子位清操作，即 $\text{GPIO\_OE} \&= \sim \text{wdata}$ | WO | 0x00000000 |

## SIO: GPIO\_HI\_OE\_CLR 寄存器

偏移量: 0x044

### 说明

GPIO32...47、QSPI IO 和 USB 引脚的输出使能清除。

对 GPIO\_HI\_OE 执行原子位清除，即  $\text{GPIO\_HI\_OE} \&= \sim \text{wdata}$

表 34.  
GPIO\_HI\_OE\_CLR  
寄存器

| 位     | 描述              | 类型 | 复位     |
|-------|-----------------|----|--------|
| 31:28 | <b>QSPI_SD</b>  | WO | 0x0    |
| 27    | <b>QSPI_CSN</b> | WO | 0x0    |
| 26    | <b>QSPI_SCK</b> | WO | 0x0    |
| 25    | <b>USB_DM</b>   | WO | 0x0    |
| 24    | <b>USB_DP</b>   | WO | 0x0    |
| 23:16 | 保留。             | -  | -      |
| 15:0  | <b>GPIO</b>     | WO | 0x0000 |

## SIO: GPIO\_OE\_XOR 寄存器

偏移量: 0x048

### 说明

GPIO0...31 输出使能异或

表 35.  
GPIO\_OE\_XOR  
寄存器

| 位    | 描述                                                  | 类型 | 复位         |
|------|-----------------------------------------------------|----|------------|
| 31:0 | 对 GPIO_OE 执行原子位异或操作，即 <code>GPIO_OE ^= wdata</code> | WO | 0x00000000 |

## SIO: GPIO\_HI\_OE\_XOR 寄存器

偏移量: 0x04c

### 说明

GPIO32...47、QSPI IO 和 USB 引脚的输出使能异或

对 GPIO\_HI\_OE 执行原子位异或操作，即 `GPIO_HI_OE ^= wdata`

表 36.  
GPIO\_HI\_OE\_XOR  
寄存器

| 位     | 描述              | 类型 | 复位     |
|-------|-----------------|----|--------|
| 31:28 | <b>QSPI_SD</b>  | WO | 0x0    |
| 27    | <b>QSPI_CSN</b> | WO | 0x0    |
| 26    | <b>QSPI_SCK</b> | WO | 0x0    |
| 25    | <b>USB_DM</b>   | WO | 0x0    |
| 24    | <b>USB_DP</b>   | WO | 0x0    |
| 23:16 | 保留。             | -  | -      |
| 15:0  | <b>GPIO</b>     | WO | 0x0000 |

## SIO: FIFO\_ST 寄存器

偏移: 0x050

### 描述

核间 FIFO（邮箱）的状态寄存器

核心0 →核心1方向有一个FIFO，核心1 →核心0方向也有一个FIFO。两个FIFO均为32位宽，深度为8字。

核心0可见1→0 FIFO (RX) 的读取端和0→1 FIFO (TX) 的写入端。

核心1可见0→1 FIFO (RX) 的读取端和1→0 FIFO (TX) 的写入端。

每个核心的SIO IRQ为其FIFO\_ST寄存器中VLD、WOF和ROE字段的逻辑或。

表37. FIFO\_ST  
寄存器

| 位    | 描述                                                  | 类型 | 复位  |
|------|-----------------------------------------------------|----|-----|
| 31:4 | 保留。                                                 | -  | -   |
| 3    | <b>ROE</b> : 指示RX FIFO为空时被读取的粘滞标志位。该读取操作被FIFO忽略。    | WC | 0x0 |
| 2    | <b>WOF</b> : 指示TX FIFO已满时被写入的粘滞标志位。此写入操作被FIFO忽略。    | WC | 0x0 |
| 1    | <b>RDY</b> : 当该核的TX FIFO未满（即FIFO_WR已准备好接收更多数据）时，值为1 | 只读 | 0x1 |
| 0    | <b>VLD</b> : 当该核的RX FIFO非空（即FIFO_RD有效）时，值为1         | 只读 | 0x0 |

## SIO: FIFO\_WR寄存器

偏移: 0x054

表38. FIFO\_WR  
寄存器

| 位    | 描述               | 类型 | 复位         |
|------|------------------|----|------------|
| 31:0 | 对本核 TX FIFO 的写访问 | WF | 0x00000000 |

## SIO: FIFO\_RD寄存器

偏移: 0x058

表39. FIFO\_RD  
寄存器

| 位    | 描述               | 类型 | 复位 |
|------|------------------|----|----|
| 31:0 | 对本核 RX FIFO 的读访问 | RF | -  |

## SIO: SPINLOCK\_ST寄存器

偏移: 0x05c

表40.  
SPINLOCK\_ST  
寄存器

| 位    | 描述                                           | 类型 | 复位         |
|------|----------------------------------------------|----|------------|
| 31:0 | 自旋锁状态<br>包含全部32个自旋锁状态的位图（1表示已锁定）。<br>主要用于调试。 | 只读 | 0x00000000 |

## SIO: INTERP0\_ACCUM0寄存器

偏移: 0x080

表 41.  
INTERP0\_ACCUM0  
寄存器

| 位    | 描述          | 类型 | 复位         |
|------|-------------|----|------------|
| 31:0 | 累加器 0 的读写访问 | 读写 | 0x00000000 |

## SIO: INTERP0\_ACCUM1 寄存器

偏移: 0x084

表 42.  
INTERP0\_ACCUM1  
寄存器

| 位    | 描述          | 类型 | 复位         |
|------|-------------|----|------------|
| 31:0 | 累加器 1 的读写访问 | 读写 | 0x00000000 |

## SIO: INTERP0\_BASE0 寄存器

偏移: 0x088

表 43.  
INTERP0\_BASE0  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 对BASE0寄存器的读写访问。 | 读写 | 0x00000000 |

## SIO: INTERP0\_BASE1 寄存器

偏移: 0x08c

表 44.  
INTERP0\_BASE1  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 对BASE1寄存器的读写访问。 | 读写 | 0x00000000 |

## SIO: INTERP0\_BASE2 寄存器

偏移: 0x090

表 45.  
INTERP0\_BASE2  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 对BASE2寄存器的读写访问。 | 读写 | 0x00000000 |

## SIO: INTERP0\_POP\_LANE0 寄存器

偏移: 0x094

表 46.  
INTERP0\_POP\_LANE0  
寄存器

| 位    | 描述                                | 类型 | 复位         |
|------|-----------------------------------|----|------------|
| 31:0 | 读取LANE0结果，同时将该通道结果写入两个累加器（POP操作）。 | 只读 | 0x00000000 |

## SIO: INTERP0\_POP\_LANE1 寄存器

偏移: 0x098

表 47.  
INTERP0\_POP\_LANE1  
寄存器

| 位    | 描述                                | 类型 | 复位         |
|------|-----------------------------------|----|------------|
| 31:0 | 读取LANE1结果，同时将该通道结果写入两个累加器（POP操作）。 | 只读 | 0x00000000 |

## SIO: INTERP0\_POP\_FULL 寄存器

偏移: 0x09c

表 48.  
INTERP0\_POP\_FULL  
寄存器

| 位    | 描述                               | 类型 | 复位         |
|------|----------------------------------|----|------------|
| 31:0 | 读取 FULL 结果，同时将通道结果写入两个累加器。（POP）。 | 只读 | 0x00000000 |

## SIO: INTERP0\_PEEK\_LANE0 寄存器

偏移: 0x0a0

表 49.  
INTERP0\_PEEK\_LANE  
0 寄存器

| 位    | 描述                           | 类型 | 复位         |
|------|------------------------------|----|------------|
| 31:0 | 读取LANE0结果，不改变任何内部状态（PEEK操作）。 | 只读 | 0x00000000 |

## SIO: INTERP0\_PEEK\_LANE1 寄存器

偏移: 0x0a4

表 50。  
INTERP0\_PEEK\_LANE  
1 寄存器

| 位    | 描述                           | 类型 | 复位         |
|------|------------------------------|----|------------|
| 31:0 | 读取LANE1结果，不改变任何内部状态（PEEK操作）。 | 只读 | 0x00000000 |

## SIO: INTERP0\_PEEK\_FULL 寄存器

偏移: 0x0a8

表 51。  
INTERP0\_PEEK\_FULL  
寄存器

| 位    | 描述                          | 类型 | 复位         |
|------|-----------------------------|----|------------|
| 31:0 | 读取FULL结果，不改变任何内部状态（PEEK操作）。 | 只读 | 0x00000000 |

## SIO: INTERP0\_CTRL\_LANE0 寄存器

偏移: 0x0ac

### 描述

通道0控制寄存器

表 52。  
INTERP0\_CTRL\_LANE  
0 寄存器

| 位     | 描述                                                                                                                                                                                                                                                                                          | 类型 | 复位  |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:26 | 保留。                                                                                                                                                                                                                                                                                         | -  | -   |
| 25    | <b>OVERF:</b> 如果 OVERF0 或 OVERF1 被置位，则该位被设置。                                                                                                                                                                                                                                                | 只读 | 0x0 |
| 24    | <b>OVERF1:</b> 指示 ACCUM1 中任何掩码屏蔽的最高有效位是否被置位。                                                                                                                                                                                                                                                | 只读 | 0x0 |
| 23    | <b>OVERF0:</b> 指示 ACCUM0 中任何掩码屏蔽的最高有效位是否被置位。                                                                                                                                                                                                                                                | 只读 | 0x0 |
| 22    | 保留。                                                                                                                                                                                                                                                                                         | -  | -   |
| 21    | <b>BLEND:</b> 仅存在于每个核心的 INTERP0 上。如果启用了 BLEND 模式：<br>- LANE1 结果为 BASE0 与 BASE1 之间的线性插值，由第 1 通道移位和掩码值的 8 个最低有效位控制（该值为 0 至 255/256 之间的分数）<br><br>- LANE0 结果不加 BASE0（仅输出第 1 通道移位加掩码值的 8 个最低有效位）<br><br>- FULL 结果不加第 1 通道的移位加掩码值（即 BASE2 + 第 0 通道的移位加掩码）<br><br>LANE1 的 SIGNED 标志控制插值结果是否为有符号数。 | 读写 | 0x0 |
| 20:19 | <b>FORCE_MSB:</b> 按位或至总线上传递给处理器的通道结果第 29:28 位<br>。<br>对内部 32 位数据路径无任何影响。便于用该通道生成<br><br>指向 Flash 或 SRAM 的指针序列。                                                                                                                                                                              | 读写 | 0x0 |
| 18    | <b>ADD_RAW:</b> 若值为 1，则绕过 LANE0 结果的掩码与移位操作。此操作不影响 FULL 结果。                                                                                                                                                                                                                                  | 读写 | 0x0 |
| 17    | <b>CROSS_RESULT:</b> 若为 1，则在数据弹出时将相反通道的结果馈入该通道的累加器。                                                                                                                                                                                                                                         | 读写 | 0x0 |
| 16    | <b>CROSS_INPUT:</b> 若设置为1，则将对向车道的累加器输入至本车道的移位与掩码硬件。<br>即使设置了ADD_RAW，该功能仍然生效（CROSS_INPUT多路复用器位于移位与掩码绕过之前）。                                                                                                                                                                                   | 读写 | 0x0 |
| 15    | <b>SIGNED:</b> 若设置SIGNED，移位并掩码后的累加器值将进行带符号扩展至32位。<br>在加至BASE0之前，且LANE0的PEEK/POP操作将视为扩展至32位。<br>由处理器读取时生效。                                                                                                                                                                                   | 读写 | 0x0 |

| 位     | 描述                                                          | 类型 | 复位   |
|-------|-------------------------------------------------------------|----|------|
| 14:10 | <b>MASK_MSB</b> : 掩码允许通过的最高有效位（包含该位）。将MSB设置小于LSB可能导致芯片功能反常。 | 读写 | 0x00 |
| 9:5   | <b>MASK_LSB</b> : 掩码允许通过的最低有效位（包含该位）。                       | 读写 | 0x00 |
| 4:0   | <b>SHIFT</b> : 对累加器执行的掩码前右旋操作。通过适当配置掩码，可合成左移与右移操作。          | 读写 | 0x00 |

## SIO: INTERP0\_CTRL\_LANE1 寄存器

偏移: 0x0b0

### 说明

通道1控制寄存器

表53。  
INTERP0\_CTRL\_LANE  
1 寄存器

| 位     | 描述                                                                                                           | 类型 | 复位   |
|-------|--------------------------------------------------------------------------------------------------------------|----|------|
| 31:21 | 保留。                                                                                                          | -  | -    |
| 20:19 | <b>FORCE_MSB</b> : 按位或至总线上传递给处理器的通道结果第 29:28 位。<br>对内部 32 位数据路径无任何影响。便于用该通道生成指向 Flash 或 SRAM 的指针序列。          | 读写 | 0x0  |
| 18    | <b>ADD_RAW</b> : 若值为 1，则跳过对 LANE1 结果的掩码与移位操作。此操作不影响 FULL 结果。                                                 | 读写 | 0x0  |
| 17    | <b>CROSS_RESULT</b> : 若为 1，则在数据弹出时将相反通道的结果馈入该通道的累加器。                                                         | 读写 | 0x0  |
| 16    | <b>CROSS_INPUT</b> : 若设置为1，则将对向车道的累加器输入至本车道的移位与掩码硬件。<br>即使设置了ADD_RAW，该功能仍然生效（CROSS_INPUT多路复用器位于移位与掩码绕过之前）。   | 读写 | 0x0  |
| 15    | <b>SIGNED</b> : 若设置SIGNED，移位并掩码后的累加器值将进行带符号扩展至32位。<br>在加至 BASE1 之前，且 LANE1 的 PEEK/POP 操作显示扩展至 32 位由处理器读取时生效。 | 读写 | 0x0  |
| 14:10 | <b>MASK_MSB</b> : 掩码允许通过的最高有效位（包含该位）。将MSB设置小于LSB可能导致芯片功能反常。                                                  | 读写 | 0x00 |
| 9:5   | <b>MASK_LSB</b> : 掩码允许通过的最低有效位（包含该位）。                                                                        | 读写 | 0x00 |
| 4:0   | <b>SHIFT</b> : 对累加器执行的掩码前右旋操作。通过适当配置掩码，可合成左移与右移操作。                                                           | 读写 | 0x00 |

## SIO: INTERP0\_ACCUM0\_ADD 寄存器

偏移: 0x0b4

表 54。  
INTERP0\_ACCUM0\_AD  
D 寄存器

| 位     | 描述                                                   | 类型 | 复位       |
|-------|------------------------------------------------------|----|----------|
| 31:24 | 保留。                                                  | -  | -        |
| 23:0  | 写入的值将以原子方式累加至ACCUM0<br>读取值为通道 0 的原始移位与掩码值（未加 BASE0）。 | 读写 | 0x000000 |

## SIO: INTERP0\_ACCUM1\_ADD 寄存器

偏移: 0x0b8

表 55。  
INTERP0\_ACCUM1\_AD  
D 寄存器

| 位     | 描述                                                   | 类型 | 复位       |
|-------|------------------------------------------------------|----|----------|
| 31:24 | 保留。                                                  | -  | -        |
| 23:0  | 写入的值将以原子方式累加至ACCUM1<br>读取值为通道 1 的原始移位与掩码值（未加 BASE1）。 | 读写 | 0x000000 |

## SIO: INTERP0\_BASE\_1AND0 寄存器

偏移: 0x0bc

表 56。  
INTERP0\_BASE\_1AND  
0 寄存器

| 位    | 描述                                                                      | 类型 | 复位         |
|------|-------------------------------------------------------------------------|----|------------|
| 31:0 | 写入时，低16位同时写入BASE0，高位写入BASE1。<br>若该通道的 SIGNED 标志被设置，则每个半部分将进行符号扩展至 32 位。 | WO | 0x00000000 |

## SIO: INTERP1\_ACCUM0 寄存器

偏移: 0x0c0

表 57。  
INTERP1\_ACCUM0  
寄存器

| 位    | 描述          | 类型 | 复位         |
|------|-------------|----|------------|
| 31:0 | 累加器 0 的读写访问 | 读写 | 0x00000000 |

## SIO: INTERP1\_ACCUM1 寄存器

偏移: 0x0c4

表 58。  
INTERP1\_ACCUM1  
寄存器

| 位    | 描述          | 类型 | 复位         |
|------|-------------|----|------------|
| 31:0 | 累加器 1 的读写访问 | 读写 | 0x00000000 |

## SIO: INTERP1\_BASE0 寄存器

偏移: 0x0c8

表 59。  
INTERP1\_BASE0  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 对BASE0寄存器的读写访问。 | 读写 | 0x00000000 |

## SIO: INTERP1\_BASE1 寄存器

偏移: 0x0cc

表 60。  
INTERP1\_BASE1  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 对BASE1寄存器的读写访问。 | 读写 | 0x00000000 |

## SIO: INTERP1\_BASE2 寄存器

偏移: 0x0d0

表 61.  
INTERP1\_BASE2  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 对BASE2寄存器的读写访问。 | 读写 | 0x00000000 |

## SIO: INTERP1\_POP\_LANE0 寄存器

偏移: 0x0d4

表 62.  
INTERP1\_POP\_LANE0  
寄存器

| 位    | 描述                                | 类型 | 复位         |
|------|-----------------------------------|----|------------|
| 31:0 | 读取LANE0结果，同时将该通道结果写入两个累加器（POP操作）。 | 只读 | 0x00000000 |

## SIO: INTERP1\_POP\_LANE1 寄存器

偏移: 0x0d8

表 63.  
INTERP1\_POP\_LANE1  
寄存器

| 位    | 描述                                | 类型 | 复位         |
|------|-----------------------------------|----|------------|
| 31:0 | 读取LANE1结果，同时将该通道结果写入两个累加器（POP操作）。 | 只读 | 0x00000000 |

## SIO: INTERP1\_POP\_FULL 寄存器

偏移: 0x0dc

表 64.  
INTERP1\_POP\_FULL  
寄存器

| 位    | 描述                               | 类型 | 复位         |
|------|----------------------------------|----|------------|
| 31:0 | 读取 FULL 结果，同时将通道结果写入两个累加器。（POP）。 | 只读 | 0x00000000 |

## SIO: INTERP1\_PEEK\_LANE0 寄存器

偏移: 0x0e0

表 65.  
INTERP1\_PEEK\_LANE  
0 寄存器

| 位    | 描述                           | 类型 | 复位         |
|------|------------------------------|----|------------|
| 31:0 | 读取LANE0结果，不改变任何内部状态（PEEK操作）。 | 只读 | 0x00000000 |

## SIO: INTERP1\_PEEK\_LANE1 寄存器

偏移: 0x0e4

表 66.  
INTERP1\_PEEK\_LANE  
1 寄存器

| 位    | 描述                           | 类型 | 复位         |
|------|------------------------------|----|------------|
| 31:0 | 读取LANE1结果，不改变任何内部状态（PEEK操作）。 | 只读 | 0x00000000 |

## SIO: INTERP1\_PEEK\_FULL 寄存器

偏移: 0x0e8

表 67.  
INTERP1\_PEEK\_FULL  
寄存器

| 位    | 描述                          | 类型 | 复位         |
|------|-----------------------------|----|------------|
| 31:0 | 读取FULL结果，不改变任何内部状态（PEEK操作）。 | 只读 | 0x00000000 |

## SIO: INTERP1\_CTRL\_LANE0 寄存器

偏移: 0x0ec

### 描述

通道0控制寄存器

表 68.  
INTERP1\_CTRL\_LANE  
0 寄存器

| 位     | 描述                                                                                                                                                 | 类型 | 复位   |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:26 | 保留。                                                                                                                                                | -  | -    |
| 25    | <b>OVERF</b> : 如果 OVERF0 或 OVERF1 被置位，则该位被设置。                                                                                                      | 只读 | 0x0  |
| 24    | <b>OVERF1</b> : 指示 ACCUM1 中任何掩码屏蔽的最高有效位是否被置位。                                                                                                      | 只读 | 0x0  |
| 23    | <b>OVERF0</b> : 指示 ACCUM0 中任何掩码屏蔽的最高有效位是否被置位。                                                                                                      | 只读 | 0x0  |
| 22    | <b>CLAMP</b> : 仅存在于每个核心的 INTERP1 上。若启用 CLAMP 模式：<br>- LANE0 结果为位移并掩码的 ACCUM0，由下限 BASE0 与上限 BASE1 限幅。<br>- 这些比较的符号属性由 LANE0_CTRL_SIGNED 决定。<br>- 符号 | 读写 | 0x0  |
| 21    | 保留。                                                                                                                                                | -  | -    |
| 20:19 | <b>FORCE_MSB</b> : 按位或至总线上传递给处理器的通道结果第 29:28 位。<br>对内部 32 位数据路径无任何影响。便于用该通道生成<br>指向 Flash 或 SRAM 的指针序列。                                            | 读写 | 0x0  |
| 18    | <b>ADD_RAW</b> : 若值为 1，则绕过 LANE0 结果的掩码与移位操作。此操作不影响 FULL 结果。                                                                                        | 读写 | 0x0  |
| 17    | <b>CROSS_RESULT</b> : 若为 1，则在数据弹出时将相反通道的结果馈入该通道的累加器。                                                                                               | 读写 | 0x0  |
| 16    | <b>CROSS_INPUT</b> : 若设置为1，则将对向车道的累加器输入至本车道的移位与掩码硬件。<br>即使设置了ADD_RAW，该功能仍然生效（CROSS_INPUT多路复用器位于移位与掩码绕过之前）。                                         | 读写 | 0x0  |
| 15    | <b>SIGNED</b> : 若设置SIGNED，移位并掩码后的累加器值将进行带符号扩展至32位。<br>在加至BASE0之前，且LANE0的PEEK/POP操作将视为扩展至32位。<br>由处理器读取时生效。                                         | 读写 | 0x0  |
| 14:10 | <b>MASK_MSB</b> : 掩码允许通过的最高有效位（包含该位）。<br>将MSB设置小于LSB可能导致芯片功能反常。                                                                                    | 读写 | 0x00 |
| 9:5   | <b>MASK_LSB</b> : 掩码允许通过的最低有效位（包含该位）。                                                                                                              | 读写 | 0x00 |
| 4:0   | <b>SHIFT</b> : 对累加器执行的掩码前右旋操作。通过适当配置掩码，可合成左移与右移操作。                                                                                                 | 读写 | 0x00 |

## SIO: INTERP1\_CTRL\_LANE1 寄存器

偏移量: 0x0f0

### 描述

通道1控制寄存器

表 69  
INTERP1\_CTRL\_LANE  
1 寄存器

| 位     | 描述                                                                                                           | 类型 | 复位   |
|-------|--------------------------------------------------------------------------------------------------------------|----|------|
| 31:21 | 保留。                                                                                                          | -  | -    |
| 20:19 | <b>FORCE_MSB</b> : 按位或至总线上传递给处理器的通道结果第 29:28 位。<br>对内部 32 位数据路径无任何影响。便于用该通道生成指向 Flash 或 SRAM 的指针序列。          | 读写 | 0x0  |
| 18    | <b>ADD_RAW</b> : 若值为 1，则跳过对 LANE1 结果的掩码与移位操作。此操作不影响 FULL 结果。                                                 | 读写 | 0x0  |
| 17    | <b>CROSS_RESULT</b> : 若为 1，则在数据弹出时将相反通道的结果馈入该通道的累加器。                                                         | 读写 | 0x0  |
| 16    | <b>CROSS_INPUT</b> : 若设置为1，则将对向车道的累加器输入至本车道的移位与掩码硬件。<br>即使设置了ADD_RAW，该功能仍然生效（CROSS_INPUT多路复用器位于移位与掩码绕过之前）。   | 读写 | 0x0  |
| 15    | <b>SIGNED</b> : 若设置SIGNED，移位并掩码后的累加器值将进行带符号扩展至32位。<br>在加至 BASE1 之前，且 LANE1 的 PEEK/POP 操作显示扩展至 32 位由处理器读取时生效。 | 读写 | 0x0  |
| 14:10 | <b>MASK_MSB</b> : 掩码允许通过的最高有效位（包含该位）。<br>将MSB设置小于LSB可能导致芯片功能反常。                                              | 读写 | 0x00 |
| 9:5   | <b>MASK_LSB</b> : 掩码允许通过的最低有效位（包含该位）。                                                                        | 读写 | 0x00 |
| 4:0   | <b>SHIFT</b> : 对累加器执行的掩码前右旋操作。通过适当配置掩码，可合成左移与右移操作。                                                           | 读写 | 0x00 |

## SIO: INTERP1\_ACCUM0\_ADD 寄存器

偏移量: 0x0f4

表 70  
INTERP1\_ACCUM0\_AD  
D 寄存器

| 位     | 描述                                                   | 类型 | 复位       |
|-------|------------------------------------------------------|----|----------|
| 31:24 | 保留。                                                  | -  | -        |
| 23:0  | 写入的值将以原子方式累加至ACCUM0<br>读取值为通道 0 的原始移位与掩码值（未加 BASE0）。 | 读写 | 0x000000 |

## SIO: INTERP1\_ACCUM1\_ADD 寄存器

偏移量: 0x0f8

表 71  
INTERP1\_ACCUM1\_AD  
D 寄存器

| 位     | 描述                                                   | 类型 | 复位       |
|-------|------------------------------------------------------|----|----------|
| 31:24 | 保留。                                                  | -  | -        |
| 23:0  | 写入的值将以原子方式累加至ACCUM1<br>读取值为通道 1 的原始移位与掩码值（未加 BASE1）。 | 读写 | 0x000000 |

## SIO: INTERP1\_BASE\_1AND0 寄存器

偏移量: 0x0fc

表 72  
INTERP1\_BASE\_1AND  
0 寄存器

| 位    | 描述                                                                      | 类型 | 复位         |
|------|-------------------------------------------------------------------------|----|------------|
| 31:0 | 写入时，低16位同时写入BASE0，高位写入BASE1。<br>若该通道的 SIGNED 标志被设置，则每个半部分将进行符号扩展至 32 位。 | WO | 0x00000000 |

## SIO: SPINLOCK0、SPINLOCK1、...、SPINLOCK30、SPINLOCK31 寄存器

偏移量：0x100、0x104、...、0x178、0x17c

表 73。SPINLOCK0、S  
PINLOCK1、...  
、SPINLOCK3  
0、SPINLOC  
K31 寄存器

| 位    | 描述                                                                                                                                      | 类型 | 复位         |
|------|-----------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 从自旋锁地址读取将：<br>- 若锁已经被锁定，返回 0<br>- 否则返回非零值，同时声明该锁<br><br>写入（任意值）将释放该锁。<br>若核心 0 与核心 1 同时尝试声明同一把锁，则以核心 0 优先。<br><br>操作成功时返回值为 0x1 << 锁编号。 | 读写 | 0x00000000 |

## SIO: DOORBELL\_OUT\_SET 寄存器

偏移量：0x180

表 74。  
DOORBELL\_OUT\_SET  
寄存器

| 位    | 描述                                                                                                                                  | 类型 | 复位   |
|------|-------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                                                                 | -  | -    |
| 7:0  | 触发对侧核心的门铃中断。<br><br>向某位写入1以设置对侧核心 DOORBELL_IN 寄存器中对应的位。此操作将引发对侧核心的门铃中断。<br><br>读取该寄存器以获取对侧核心当前激活的门铃状态。这相当于该核心读取其自身的 DOORBELL_IN 状态。 | 读写 | 0x00 |

## SIO: DOORBELL\_OUT\_CLR 寄存器

偏移量：0x184

表 75。  
DOORBELL\_OUT\_CLR  
寄存器

| 位    | 描述                                                                                                                                                                                                                                                           | 类型 | 复位   |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                                                                                                                                                                                          | -  | -    |
| 7:0  | <p>清除发送至对核的门铃信号。该寄存器用于调试及初始化目的。</p> <p>向 DOORBELL_OUT_CLR 中的某个位写入 1 会清除对侧核心 DOORBELL_IN 中对应的位。清除所有位将导致该核心的门铃中断撤销。鉴于软件通常先通过 DOORBELL_OUT_SET 发送事件，再通过向 DOORBELL_IN_CLR 写入进行事件确认，因此应谨慎使用此寄存器，以避免竞态条件。</p> <p>读取返回另一核心当前已触发的门铃状态，即该核心读取其自身的 DOORBELL_IN 状态。</p> | WC | 0x00 |

## SIO: DOORBELL\_IN\_SET 寄存器

偏移: 0x188

表 76。  
DOORBELL\_IN\_SET  
寄存器

| 位    | 描述                                   | 类型 | 复位   |
|------|--------------------------------------|----|------|
| 31:8 | 保留。                                  | -  | -    |
| 7:0  | 写入 1 以触发本核心的门铃中断。读取以获取本核心当前已触发的门铃状态。 | 读写 | 0x00 |

## SIO: DOORBELL\_IN\_CLR 寄存器

偏移: 0x18c

表 77。  
DOORBELL\_IN\_CLR  
寄存器

| 位    | 描述                                                                                                          | 类型 | 复位   |
|------|-------------------------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                                         | -  | -    |
| 7:0  | <p>检查并确认发送至该核心的门铃信号。当本寄存器中任一位为1时，该核心的门铃中断即被触发。</p> <p>写入1至每个位以清除此位。全部位清除后，门铃中断即取消触发。读取以获取本核心当前已触发的门铃状态。</p> | WC | 0x00 |

## SIO: PERI\_NONSEC 寄存器

偏移: 0x190

### 描述

将某些核心本地外设从安全SIO断开，并连接至非安全SIO，以使非安全软件得以使用。当外设连接到非安全SIO时，若从安全SIO访问该外设，反之亦然，均会导致总线错误。

此寄存器为每核心专属，且仅存在于安全SIO内。

大多数 SIO 硬件在安全与非安全 SIO 之间是复制的，因此未在此寄存器中列出。

表 78。  
PERI\_NONSEC  
寄存器

| 位    | 描述                                                          | 类型 | 复位  |
|------|-------------------------------------------------------------|----|-----|
| 31:6 | 保留。                                                         | -  | -   |
| 5    | <b>TMDS:</b> 当值为 1 时，断开本核心 TMDS 编码器与安全 SIO 的连接，并连接至非安全 SIO。 | 读写 | 0x0 |

| 位   | 描述                                                           | 类型 | 复位  |
|-----|--------------------------------------------------------------|----|-----|
| 4:2 | 保留。                                                          | -  | -   |
| 1   | <b>INTERP1</b> : 当值为 1 时，断开本核心插值器 1 与安全 SIO 的连接，且连接至非安全 SIO。 | 读写 | 0x0 |
| 0   | <b>INTERP0</b> : 如果为1，则将该核心的插值器0从安全SIO中分离，且连接至非安全 SIO。       | 读写 | 0x0 |

## SIO: RISCV\_SOFTIRQ寄存器

偏移量: 0x1a0

### 描述

控制RISC-V核心上标准软件中断（MIP.MSIP）的触发与否。

与RISC-V定时器不同，此中断未被路由至标准系统级中断线，因此Arm核心无法使用。

两个核心在同一周期内写入该寄存器是安全的。置位/清除效果在两个核心之间累积后应用。若某标志在同一周期既被置位又被清除，则仅置位生效。

表79。  
RISCV\_SOFTIRQ  
Q寄存器

| 位     | 描述                                                         | 类型 | 复位  |
|-------|------------------------------------------------------------|----|-----|
| 31:10 | 保留。                                                        | -  | -   |
| 9     | <b>CORE1_CLR</b> : 写入1以原子方式清除核心1的软件中断标志。<br>读取该寄存器以获取标志状态。 | 读写 | 0x0 |
| 8     | <b>CORE0_CLR</b> : 写入1以原子方式清除核心0的软件中断标志。<br>读取该寄存器以获取标志状态。 | 读写 | 0x0 |
| 7:2   | 保留。                                                        | -  | -   |
| 1     | <b>CORE1_SET</b> : 写入1以原子方式设置核心1的软件中断标志。读取该寄存器以获取标志状态。     | 读写 | 0x0 |
| 0     | <b>CORE0_SET</b> : 写入1以原子方式设置核心0的软件中断标志。读取该寄存器以获取标志状态。     | 读写 | 0x0 |

## SIO: MTIME\_CTRL寄存器

偏移量: 0x1a4

### 描述

RISC-V 64位机器模式计时器控制寄存器。该计时器仅存在于安全SIO中，因此仅可由处于安全模式的Arm核心或处于机器模式的RISC-V核心访问。

注意，尽管该计时器遵循RISC-V特权规范，但同样适用于Arm核心。中断被路由至普通系统级中断线及RISC-V核心的MIP.MTIP输入。

表80。  
MTIME\_CTRL寄存器

| 位    | 描述                                                                    | 类型 | 复位  |
|------|-----------------------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                                   | -  | -   |
| 3    | <b>DBGPAUSE_CORE1</b> : 如果为1，当核心1处于调试停止状态时，定时器将暂停。                    | 读写 | 0x1 |
| 2    | <b>DBGPAUSE_CORE0</b> : 若值为1，当核心0处于调试暂停状态时，计时器将暂停。                    | 读写 | 0x1 |
| 1    | <b>FULLSPEED</b> : 若值为1，则计时器在每个时钟周期递增（即直接通过系统时钟运行），而非在系统级计时器滴答输入信号递增。 | 读写 | 0x0 |

| 位 | 描述                        | 类型 | 复位  |
|---|---------------------------|----|-----|
| 0 | EN：计时器使能位。值为0时，计时器不会自动递增。 | 读写 | 0x1 |

## SIO: MTIME 寄存器

偏移: 0x1b0

表81. MTIME 寄存器

| 位    | 描述                                                         | 类型 | 复位         |
|------|------------------------------------------------------------|----|------------|
| 31:0 | 对RISC-V机器模式计时器高半部的读写访问权限。该寄存器由两个核心共享。若两个核心在同一周期内写入，则核心1优先。 | 读写 | 0x00000000 |

## SIO: MTIMEH 寄存器

偏移: 0x1b4

表82. MTIMEH 寄存器

| 位    | 描述                                                         | 类型 | 复位         |
|------|------------------------------------------------------------|----|------------|
| 31:0 | 对RISC-V机器模式计时器高半部的读写访问权限。该寄存器由两个核心共享。若两个核心在同一周期内写入，则核心1优先。 | 读写 | 0x00000000 |

## SIO: MTIMECMP 寄存器

偏移: 0x1b8

表83. MTIMECMP 寄存器

| 位    | 描述                                                                                                                                  | 类型 | 复位         |
|------|-------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | RISC-V 机器模式定时器比较器的低半部分。此寄存器为核心本地寄存器，即每个核心均有该寄存器副本，比较结果路由至对应的中断线路。<br><br>当 MTIME 大于或等于 MTIMECMP 时，将产生定时器中断。该比较为无符号比较，基于完整的 64 位值执行。 | 读写 | 0xffffffff |

## SIO: MTIMECMPPH 寄存器

偏移: 0x1bc

表84. MTIMECMPPH 寄存器

| 位    | 描述                                                                                                       | 类型 | 复位         |
|------|----------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | RISC-V 机器模式定时器比较器的高半部分。该寄存器为核心本地寄存器。<br><br>当 MTIME 大于或等于 MTIMECMPPH 时，将产生定时器中断。该比较为无符号比较，基于完整的 64 位值执行。 | 读写 | 0xffffffff |

## SIO: TMDS\_CTRL寄存器

偏移: 0x1c0

### 描述

TMDS 编码器的控制寄存器。

表85. TMDS\_CTRL寄存器

| 位     | 描述                                                                                                                                                                                                  | 类型 | 复位  |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:29 | 保留。                                                                                                                                                                                                 | -  | -   |
| 28    | <b>CLEAR_BALANCE</b> : 清除TMDS编码器当前的直流平衡状态。<br>该位应在每条扫描线开始时写入一次。                                                                                                                                     | SC | 0x0 |
| 27    | <b>PIX2_NOSSHIFT</b> : 在一个周期内编码两个像素符号（读取PEEK/POP_DOUBLE寄存器时），第二个编码器会看到颜色数据寄存器的偏移版本。<br><br>此控制禁止该偏移，使两个编码器层看到相同的像素数据。该功能用于像素倍增。                                                                     | 读写 | 0x0 |
| 26:24 | <b>PIX_SHIFT</b> : 每次读取POP别名寄存器时，应用于颜色数据寄存器的偏移量。<br><br>从POP_SINGLE寄存器读取或在设置PIX2_NOSSHIFT进行像素倍增时，从POP_DOUBLE寄存器读取时，将产生指示的偏移。<br><br>当PIX2_NOSSHIFT清除时，从POP_DOUBLE寄存器读取数据将按指示位移量的两倍移位。（移位32表示不进行移位。） | 读写 | 0x0 |
|       | 枚举值：                                                                                                                                                                                                |    |     |
|       | 0x0 → 0: 不移位颜色数据寄存器。                                                                                                                                                                                |    |     |
|       | 0x1 → 1: 颜色数据寄存器左移1位。                                                                                                                                                                               |    |     |
|       | 0x2 → 2: 颜色数据寄存器左移2位。                                                                                                                                                                               |    |     |
|       | 0x3 → 4: 颜色数据寄存器左移4位。                                                                                                                                                                               |    |     |
|       | 0x4 → 8: 颜色数据寄存器左移8位。                                                                                                                                                                               |    |     |
|       | 0x5 → 16: 颜色数据寄存器左移16位。                                                                                                                                                                             |    |     |
| 23    | <b>INTERLEAVE</b> : 启用PEEK_SINGLE/POP_SINGLE读取的通道交错。<br><br>当禁用交错时，3个符号各作为连续的10位字段出现，通道0为最低有效位，起始于寄存器的第0位。<br><br>启用交错时，符号被打包为5个块，每块包含3条通道乘以2位（二进制位）（共计30位）。每个块包含每条通道的TMDS符号两位，其中通道0为最低有效位。         | 读写 | 0x0 |
| 22:21 | 保留。                                                                                                                                                                                                 | -  | -   |
| 20:18 | <b>L2_NBITS</b> : 通道2有效颜色最高有效位的数量（1-8位，编码为0至7）。<br>旋转后，剩余的最低有效位将被屏蔽为0。                                                                                                                              | 读写 | 0x0 |
| 17:15 | <b>L1_NBITS</b> : 通道1有效颜色最高有效位的数量（1-8位，编码为0至7）。<br>旋转后，剩余的最低有效位将被屏蔽为0。                                                                                                                              | 读写 | 0x0 |
| 14:12 | <b>L0_NBITS</b> : 通道0有效颜色最高有效位的数量（1-8位，编码为0至7）。<br>旋转后，剩余的最低有效位将被屏蔽为0。                                                                                                                              | 读写 | 0x0 |

| 位    | 描述                                                                                                                                                    | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 11:8 | <p><b>L2_ROT:</b> 对颜色累加器的16位最低有效位进行0至15位的右旋转，使通道2（红色）颜色数据的最高有效位与8位编码器输入的最高有效位对齐。</p> <p>例如，对于RGB565（红色为最高有效位），红色位于位15:11，因此应右旋转8位，以与编码器输入的位7:3对齐。</p> | 读写 | 0x0 |
| 7:4  | <p><b>L1_ROT:</b> 将颜色累加器的最低16位右旋转0-15位，使通道1（绿色）颜色数据的最高有效位与8位编码器输入的最高有效位对齐。</p> <p>例如，对于RGB565，绿色位于位10:5，因此应右旋转3位，以与编码器输入的位7:2对齐。</p>                  | 读写 | 0x0 |
| 3:0  | <p><b>L0_ROT:</b> 将颜色累加器的最低16位右旋转0-15位，使通道0（蓝色）颜色数据的最高有效位与8位编码器输入的最高有效位对齐。</p> <p>例如，对于RGB565（红色为最高有效位），蓝色位于位4:0，因此应向右旋转13位以对齐编码器输入的第7至第3位。</p>       | 读写 | 0x0 |

## SIO: TMDS\_WDATA 寄存器

偏移：0x1c4

表86。  
TMDS\_WDATA  
寄存器

| 位    | 描述                   | 类型 | 复位         |
|------|----------------------|----|------------|
| 31:0 | 对 TMDS 颜色数据寄存器的只写访问。 | WO | 0x00000000 |

## SIO: TMDS\_PEEK\_SINGLE 寄存器

偏移：0x1c8

表87。  
TMDS\_PEEK\_SINGLE  
寄存器

| 位    | 描述                                                                                                | 类型 | 复位         |
|------|---------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | <p>获取一像素颜色数据的编码，封装为32位数值（3个10位符号）。</p> <p>PEEK别名读取时不会移动颜色寄存器，但仍会推进各编码器的运行直流平衡状态。此功能对像素倍增非常有用。</p> | RF | 0x00000000 |

## SIO: TMDS\_POP\_SINGLE 寄存器

偏移：0x1cc

表88。  
TMDS\_POP\_SINGLE  
寄存器

| 位    | 描述                                                                                                                                                 | 类型 | 复位         |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取一个像素的颜色数据编码，打包成32位数值。打包格式为5个块，每块包含3条通道，每条通道2位（共计30位）。每个块包含每条通道对应TMDS符号的两位。该格式设计用于RP2350上的HSTX外设进行移位输出。<br><br>POP别名在读取时会移位颜色寄存器，并推进每个编码器的直流平衡状态。 | RF | 0x00000000 |

## SIO: TMDS\_PEEK\_DOUBLE\_L0 寄存器

偏移：0x1d0

表 89。  
TMDS\_PEEK\_DOUBLE\_L0  
寄存器

| 位    | 描述                                                                                                                             | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取两个像素颜色数据编码的通道 0。两个10位的TMDS符号被打包在32位字的低位。<br><br>PEEK别名读取时不移位颜色寄存器，但仍然会推进第0通道的直流平衡状态。当需要一次性读取所有3条通道的编码，而非逐条通道处理整行扫描时，此功能尤为有用。 | RF | 0x00000000 |

## SIO: TMDS\_POP\_DOUBLE\_L0 寄存器

偏移：0x1d4

表 90。  
TMDS\_POP\_DOUBLE\_L0  
寄存器

| 位    | 描述                                                                                               | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取两个像素颜色数据编码的通道 0。两个10位的TMDS符号被打包在32位字的低位。<br><br>POP别名在读取时根据PIX_SHIFT和PIX2_NOSHIFT的值对颜色寄存器进行偏移。 | RF | 0x00000000 |

## SIO: TMDS\_PEEK\_DOUBLE\_L1 寄存器

偏移：0x1d8

表 91。  
TMDS\_PEEK\_DOUBLE\_L1  
寄存器

| 位    | 描述                                                                                                                             | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取两个像素颜色数据编码的通道 1。两个10位的TMDS符号被打包在32位字的低位。<br><br>PEEK别名在读取时不偏移颜色寄存器，但仍会推进第1通道的DC平衡状态。当需要一次性读取所有3条通道的编码，而非逐条通道处理整行扫描时，此功能尤为有用。 | RF | 0x00000000 |

## SIO: TMDS\_POP\_DOUBLE\_L1 寄存器

偏移：0x1dc

表 92。  
TMDS\_POP\_DOUBLE\_L1  
1 寄存器

| 位    | 描述                                                                                               | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取两个像素颜色数据编码的通道 1。两个10位的TMDS符号被打包在32位字的低位。<br><br>POP别名在读取时根据PIX_SHIFT和PIX2_NOSHIFT的值对颜色寄存器进行偏移。 | RF | 0x00000000 |

## SIO: TMDS\_PEEK\_DOUBLE\_L2 寄存器

偏移：0x1e0

表 93。  
TMDS\_PEEK\_DOUBLE\_L2  
L2 寄存器

| 位    | 描述                                                                                                                             | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取两个像素颜色数据编码的第2通道。两个10位的TMDS符号被打包在32位字的低位。<br><br>PEEK别名在读取时不偏移颜色寄存器，但仍会推进第2通道的DC平衡状态。当需要一次性读取所有3条通道的编码，而非逐条通道处理整行扫描时，此功能尤为有用。 | RF | 0x00000000 |

## SIO: TMDS\_POP\_DOUBLE\_L2 寄存器

偏移量：0x1e4

表 94。  
TMDS\_POP\_DOUBLE\_L2  
2 寄存器

| 位    | 描述                                                                                               | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 获取两个像素颜色数据编码的第2通道。两个10位的TMDS符号被打包在32位字的低位。<br><br>POP别名在读取时根据PIX_SHIFT和PIX2_NOSHIFT的值对颜色寄存器进行偏移。 | RF | 0x00000000 |

## 3.2. 中断

每个内核配备了一个内部中断控制器，具有52个中断输入。大多数情况下，每个内核的中断路由完全相同，但存在一些例外，称为内核本地中断，即每个内核拥有单独的中断源，映射到各内核相同的中断编号：

- 跨内核FIFO中断：[SIO\\_IRQ\\_FIFO](#)和[SIO\\_IRQ\\_FIFO\\_NS](#)（第3.1.5节）
- 跨内核门铃中断：[SIO\\_IRQ\\_BELL](#)和[SIO\\_IRQ\\_BELL\\_NS](#)（第3.1.6节）
- RISC-V平台定时器（Arm内核亦可使用）：[SIO\\_IRQ\\_MTIMECMP](#)（第3.1.8节）
- GPIO中断：[IO\\_IRQ\\_BANK0](#)、[IRQ\\_IO\\_BANK0\\_NS](#)、[IO\\_IRQ\\_QSPI](#)、[IO\\_IRQ\\_QSPI\\_NS](#)（第9.5节）

剩余的中断输入在两个核心上具有完全相同的中断源镜像。非核心本地的中断应仅在单个核心的中断控制器中启用，且由启用该中断控制器的核心进行服务。

表 95。系统级  
中断编号。  
所有中断均被  
路由至两个  
处理器。

| IRQ | 中断源                          | IRQ | 中断源                       | IRQ | 中断源                             | IRQ | 中断源                          | IRQ | 中断源                              |
|-----|------------------------------|-----|---------------------------|-----|---------------------------------|-----|------------------------------|-----|----------------------------------|
| 0   | <a href="#">TIMER0_IRQ_0</a> | 11  | <a href="#">DMA_IRQ_1</a> | 22  | <a href="#">IO_IRQ_BANK0_NS</a> | 33  | <a href="#">UART0_IRQ</a>    | 44  | <a href="#">POWMAN_IRQ_POW</a>   |
| 1   | <a href="#">TIMER0_IRQ_1</a> | 12  | <a href="#">DMA_IRQ_2</a> | 23  | <a href="#">IO_IRQ_QSPI</a>     | 34  | <a href="#">UART1_IRQ</a>    | 45  | <a href="#">POWMAN_IRQ_TIMER</a> |
| 2   | <a href="#">TIMER0_IRQ_2</a> | 13  | <a href="#">DMA_IRQ_3</a> | 24  | <a href="#">IO_IRQ_QSPI_NS</a>  | 35  | <a href="#">ADC_IRQ_FIFO</a> | 46  | <a href="#">SPAREIRQ_IRQ_0</a>   |

| IRQ | 中断源            | IRQ | 中断源          | IRQ | 中断源              | IRQ | 中断源           | IRQ | 中断源            |
|-----|----------------|-----|--------------|-----|------------------|-----|---------------|-----|----------------|
| 3   | TIMER0_IRQ_3   | 14  | USBCTRL_IRQ  | 25  | SIO_IRQ_FIFO     | 36  | I2C0_IRQ      | 47  | SPAREIRQ_IRQ_1 |
| 4   | TIMER1_IRQ_0   | 15  | PIO0_IRQ_0   | 26  | SIO_IRQ_BELL     | 37  | I2C1_IRQ      | 48  | SPAREIRQ_IRQ_2 |
| 5   | TIMER1_IRQ_1   | 16  | PIO0_IRQ_1   | 27  | SIO_IRQ_FIFO_NS  | 38  | OTP_IRQ       | 49  | SPAREIRQ_IRQ_3 |
| 6   | TIMER1_IRQ_2   | 17  | PIO1_IRQ_0   | 28  | SIO_IRQ_BELL_NS  | 39  | TRNG_IRQ      | 50  | SPAREIRQ_IRQ_4 |
| 7   | TIMER1_IRQ_3   | 18  | PIO1_IRQ_1   | 29  | SIO_IRQ_MTIMECMP | 40  | PROC0_IRQ_CTI | 51  | SPAREIRQ_IRQ_5 |
| 8   | PWM_IRQ_WRAP_0 | 19  | PIO2_IRQ_0   | 30  | CLOCKS_IRQ       | 41  | PROC1_IRQ_CTI |     |                |
| 9   | PWM_IRQ_WRAP_1 | 20  | PIO2_IRQ_1   | 31  | SPI0_IRQ         | 42  | PLL_SYS_IRQ   |     |                |
| 10  | DMA_IRQ_0      | 21  | IO_IRQ_BANK0 | 32  | SPI1_IRQ         | 43  | PLL_USB_IRQ   |     |                |

在RP2350上，仅较低46个IRQ信号连接到系统级中断源，IRQ 46至51则硬连线为零（永不触发）。这六个备用中断，在表中标注为SPAREIRQ\_IRQ\_0至SPAREIRQ\_IRQ\_5，专门预留给核心通过Arm NVIC\_ISPR0寄存器或Hazard3 MEIIFA CSR自中断使用，例如当中断处理程序希望调度“底半部”处理程序，以在退出中断处理程序但返回前台代码前完成必须执行的工作时。

硬件支持嵌套中断：较低优先级的中断可被较高优先级的中断或故障抢占，并在较高优先级处理程序返回后恢复执行。抢占优先级顺序由中断优先级寄存器确定，起始于 NVIC\_IPR0 (Cortex-M33) 或 MEIPRA 中断优先级数组 CSR (Hazard3)。

当存在多个具有相同动态优先级的中断可被触发时，将以 IRQ 号最低的中断作为决胜者。系统级 IRQ 编号设计原则通常是将高优先级中断放置于较低的 IRQ 号，尽管实际优先级通常依赖具体应用。

### 3.2.1. 不可屏蔽中断 (NMI)

系统 IRQ 信号可通过在 NMI\_MASK0 或 NMI\_MASK1 中设置相应 IRQ 号的位，路由至 Cortex-M33 的不可屏蔽中断 (NMI) 输入。不可屏蔽中断忽略处理器的中断使能/禁用状态 (PRIMASK)，可抢占任何其他正在执行的中断。非屏蔽中断 (NMI) 通常用于需要处理器无条件关注的紧急情况，例如 PLL 锁失或电源完整性异常。

NMI 掩码寄存器为核心本地寄存器，因此每个核心可以路由不同组合的中断至其 NMI 输入端。NMI 掩码及所有其他 EPP B 寄存器会在该核心执行热复位时重置。此措施避免了 RP2040 在处理器复位后可能遗留 NMI 掩码设置的问题。

除系统级中断外，当冗余协处理器 (RCP，第 3.6.3 节) 完整性检查失败时，非屏蔽中断将被断言。此行为不可禁用，但在正常电压、频率及温度条件下，正确编程的 RCP 不会触发此中断。同理，若用户代码未执行任何 RCP 指令，RCP 将永不触发中断。当完整性检查失败时，RCP 的 NMI 输出将在两个核心上断言，并通过热复位取消断言。

### 3.2.2. 中断的进一步阅读

本节描述系统级中断请求在处理器子系统中的路由。本节省略了诸如处理器接收中断时的响应机制及处理器如何选择订阅系统级中断请求等重要细节。以下摘录了与这些主题相关的部分信息：

- 第3.7.2.5节介绍了Cortex-M33内部中断控制器NVIC。
- 从NVIC\_ISER0开始的寄存器列表描述了NVIC操作的控制寄存器。
- 第3.7.4.6节为Cortex-M33异常处理提供了概述。

- Armv8-M架构参考手册详述了异常处理的架构规则。
- 第3.8.4节描述了标准RISC-V陷阱处理机制。
- 第3.8.4.2节介绍了标准RISC-V的外部中断、定时器中断及软件中断请求，以及它们在RP2350上的连接方式。
- 第3.8.6.1节介绍了Xh3irq中断控制器，该控制器为Hazard3上的系统级中断提供优先级控制支持。
- 每个外设都有其各自的中断寄存器，用于控制其系统级中断的触发，详见表95—更多信息请参阅外设文档

### 3.3. 事件信号 (Arm)

通过使用 `WFE` 指令，Cortex-M33可进入睡眠状态，直到发生“事件”（或中断）它还可使用 `SEV` 指令产生事件RP2350在两个处理器之间交叉传递事件信号：一个处理器发送的事件将在另一个处理器接收

#### 注意

事件标志为“粘滞”状态：若两个处理器同时发送事件（`SEV`）后进入睡眠状态（`WFE`），将会立即被唤醒该机制防止处理器在此情况下陷入无法唤醒的睡眠状态

当因其他主设备写操作导致其保留失效时，处理器将依照Armv8-M架构要求，接收来自全局监视器的事件信号

在 `WFE`（或 `WFI`）睡眠状态下，处理器关闭内部时钟门控以降低功耗当两个处理器均处于睡眠状态且DMA处于非活动状态时，整个RP2350可进入睡眠状态，禁用未使用的基础设施时钟，例如总线结构。任一处理器唤醒时，其余RP2350将自动唤醒。详见第6.5.2节。

### 3.4. 事件信号 (RISC-V)

Hazard3 `h3.block` 指令会暂停处理器执行，直到接收到解除阻塞信号。`h3.unblock` 指令向其他处理器发送解除阻塞信号。这些与NOP兼容的提示指令载于第3.8.6.3节。

在RP2350上，Hazard3解除阻塞输入/输出信号在两个处理器之间交叉连接，每个处理器的解除阻塞输出也反馈至其输入端。当核心因其他核心或系统DMA访问而失去预留权时，全局监视器也会向该核心发送解除阻塞信号。

Hazard3 MSLEEP CSR定义了处理器执行 `h3.block` 指令时进入的睡眠深度。默认情况下，这只是一个简单的流水线停滞，但处理器也可以自行门控时钟，并与时钟模块（第6.5.2节）协商系统级时钟的唤醒/睡眠状态。

`h3.unblock` 指令是“粘滞”的：如果处理器自上一次执行`h3.block`指令以来接收到任何`unblock`信号，则任何`h3.block`指令会立即生效。

### 3.5. 调试

串行线调试（SWD）总线提供对硬件和软件调试功能的访问，包括：

- 将固件加载到SRAM或外部闪存

- 处理器执行控制：运行/停止、单步执行、设置断点及其他标准调试功能
- 访问处理器体系结构状态
- 通过系统总线访问内存及内存映射IO
- 配置CoreSight跟踪硬件（仅限Arm处理器）

SWD总线通过两个专用引脚SWCLK和SWDIO暴露。SWCLK和SWDIO的引脚定义见表1430；其规格的详细信息见表1440。

单个SW-DP通过外部SWCLK和SWDIO引脚访问RP2350的调试子系统。该DP支持多点连接，但多点SWD并非强制使用。调试子系统中的所有硬件（RP-AP除外）亦可通过自托管调试窗口（起始地址为CORESIGHT\_PERIPH\_BASE）直接从系统总线访问。

图10. RP2350调试拓扑图。SW-DP将外部SWD引脚连接至内部调试硬件。只读存储器表列出了调试组件，用于自动发现。

AHB-AP提供对Arm处理器的调试访问，APB-AP则提供对标准RISCV调试模块的访问。  
RP-AP提供树莓派特定的控制功能，如紧急复位和调试密钥输入。  
剩余组件用于Arm跟踪。

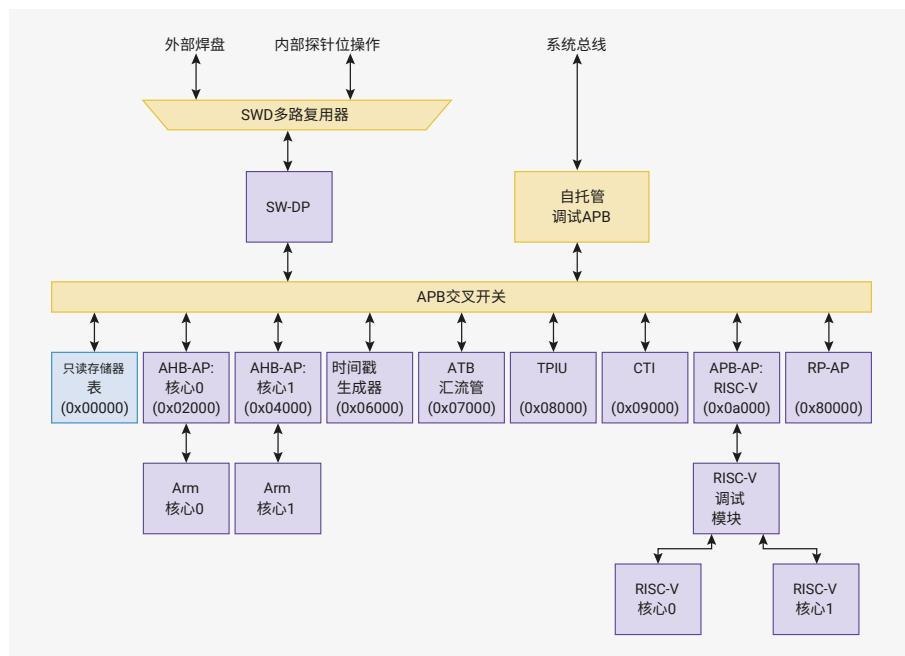


图10中括号内的数字表示调试组件在调试地址空间中的地址。

这些地址对应写入SW-DP SELECT寄存器的值（用于SWD访问），或是自托管调试访问时相对于CORESIGHT\_PERIPH\_BASE的偏移量。所有调试端口（AP）均可通过软件调试端口（SW-DP）访问，且除RP-AP外，所有AP也可通过自托管调试访问。

软件调试端口（SW-DP）和RP-AP位于始终通电的电源域，一旦外部电源接通且上电复位（POR）时间已过即开始可用。图10中的所有其他调试端口仅在满足以下条件后方可使用：

1. 电源管理器（POWMAN）已完成切换内核域的首次加电序列
2. OTP电源管理状态机（PSM）已从只读存储器（OTP）读取关键硬件配置标志
3. 系统时钟（clk\_sys）正在运行

### 3.5.1. 连接至 SW-DP

软件调试端口（SW-DP）在上电或外部复位信号（RUN引脚）断言时默认处于休眠状态。开始串行线调试（SWD）操作前，必须先执行休眠态到SWD态的转换序列。有关休眠态与SWD态切换的详细信息，请参阅Arm调试接口规范第6版：<https://developer.arm.com/documentation/ihi0074/latest/>

上电后，可使用以下序列连接到软件调试端口（SW-DP）：

1. 至少进行8个SWCLK周期，且SWDIO保持高电平。

2. 128位选择警报序列: `0x19bc0ea2, 0xe3ddafe9, 0x86852d95, 0x6209f392`, 最低有效位优先。
3. 进行4个SWCLK周期, 且SWDIO保持低电平。
4. SWD激活码序列: `0x1a`, 最低有效位优先。
5. 至少进行50个SWCLK周期, 且SWDIO保持高电平(线路复位)。
6. 读取DPIDR以退出复位状态。

为使系统从低功耗(P1.x)状态唤醒, 请设置DP CTRL/STAT寄存器中的CDBGWRUPREQ位, 随后轮询同一寄存器中的CDBGWRUPACK位, 直到该位被置位。在低功耗状态下, 仅可访问SW-DP和RP-AP, 因其余调试逻辑未通电。

### 3.5.2. Arm 调试

在调试地址空间偏移量 `0x02000` 和 `0x04000` 处有两个 AHB5 Mem-AP, 用于调试两颗Arm Cortex-M33处理器。每个 Mem-AP 是一个 AHB5 管理器, 可以访问 32 位下游地址空间。该地址空间与处理器的加载/存储指令所访问的地址空间相同, 包含系统级硬件(如内存和外设)以及处理器私有外设总线(PPB)上的处理器内部硬件。部分 PPB 寄存器仅在通过 Mem-AP 访问时可见, 处理器上运行的软件访问时则不可见。

AHB5 Mem-AP 自身的寄存器映射由 Arm 的 ADIV6 规范定义。通常此内容仅对自行实现调试转换器的人员有意义, Mem-AP 可简化理解为 DP(例如 RP2350 的 SW-DP)与下游地址空间之间的桥接器。

用于调试运行于 Cortex-M33 上软件的标准 Arm 调试寄存器, 可在 Armv8-M 架构参考手册或由 Arm Ltd. 提供的 Cortex-M33 技术参考手册中查阅。本数据手册第 3.7.5 节亦对核心内部寄存器进行了说明。

Mem-AP 可以在与处理器上运行的软件访问系统外设和内存完全相同的地址中进行访问。然而, Mem-AP 访问的权限和安全性可能不同于处理器停止时运行的软件的安全状态: Mem-AP 访问的权限和安全性通过其控制和状态字(CSW)寄存器进行明确配置。调试访问 SIO 的非安全软件时必须谨慎, 例如, 默认情况下调试器可能会访问 SIO 的安全别名, 而非软件实际访问的非安全别名。

由 ACCESSCTRL 总线访问权限寄存器(第 10.6.2 节)配置的总线过滤器将 Mem-AP 发起的总线访问与处理器上运行的软件发起的总线访问视为不同访问。这意味着可以禁止软件访问某外设, 同时仍允许调试器访问。

### 3.5.3. RISC-V 调试

调试地址空间偏移 `0x0a000` 处有一个单一的 APB Mem-AP, 仅提供对 RISC-V 调试模块(DM)的访问。DM 是调试器用于枚举系统中存在的 RISC-V hart、调试运行于各 hart 上的软件以及访问系统总线的标准组件。该组件定义于 RISC-V 调试规范中, RP2350 实现了该规范的 0.13.2 版本。

从 RISC-V 调试规范的视角来看, SW-DP 与 APB Mem-AP 共同构成系统的调试传输模块(Debug Transport Module)。DM 位于 APB-AP 下游地址空间的偏移 `0x0` 处, 其寄存器为字长大小且按字节寻址, 这意味着调试规范中的 DM 寄存器地址须乘以 4, 方能获得正确的 APB 地址。

在 RP2350 中, 每个核心仅拥有一个硬件线程(hart)。核心 0 的 hart ID 为 0, 核心 1 的 hart ID 为 1。这些 hart ID 与 DM 中所使用的 hart 索引对应。此 DM 亦配备了 hart 数组掩码选择扩展, 允许多核心同时执行重置、暂停或恢复操作。

调试模块(DM)配备了系统总线访问(SBA)扩展, 允许调试器在不停止任何核心的情况下访问系统总线。这可用于类似 Segger RTT 的最小侵入性调试技术。SBA 访问

与核心 1 的加载/存储端口竞争访问系统总线，但在总线过滤（第 10.6.2 节）中被视为与核心 1 的访问不同，这意味着可以在保留调试访问的同时锁定软件对外设的访问权限。调试模式下处理器的加载/存储同样被视为调试访问，用于总线过滤。

DM 能通过`dmcontrol.hartreset`控制单独重置各核心。仅重置所选的处理器。`dmcontrol.ndmreset`则仅重置两个处理器，这是 RISC-V 调试规范的最低要求。可通过 SW-DP 中的 SYSRESETREQ 控制执行包含 DM 在内的全系统复位；也可通过 POWMAN 配置并由看门狗触发的切换核心域复位，或任何全系统复位，如 RUN 引脚触发。看门狗触发的 PSM 复位能复位几乎所有系统级硬件，但不包括 DM；需注意的是，系统时钟的时钟发生器复位时，DM 会暂时无法访问，这也是`dmcontrol.ndmreset`仅重置处理器的原因。

关于 RISC-V 调试处理器部分的详细信息，请参见第 3.8.5 节。另请参阅 Hazard3 源代码，地址为 [github.com/Wren6991/Hazard3](https://github.com/Wren6991/Hazard3)，其中包含位于 `hdl/debug/dm/` 目录下的 DM 实现。

### 3.5.4. 调试电源域

SW-DP 和 RP-AP 位于始终通电的电源域中。这意味着即使系统处于最低功耗状态，且切换核心域（包含处理器）完全断电时，它们仍然可用。

其余调试硬件位于切换核心域中。该电源域与处理器及系统外设相同。

在 SW-DP 的 CTRL/STAT 寄存器中设置 CDBGPWRUPREQ 位，将强制切换核心域上电，使其余调试硬件变得可用。该上电过程需要一定时间，由 32 kHz 低功耗振荡器（第 8.4 节）进行序列控制，因此必须轮询 CDBGPWRUPACK 位，等待系统完成上电，方可尝试访问除 RP-AP 之外的任何 AP。有关 SW-DP 寄存器列表，请参阅 Arm 的 ADIV6 规范。

请注意，RP-AP 无需断言 CDBGPWRUPREQ 即可访问，因为它始终供电。

### 3.5.5. SWD 引脚的软件控制

SYSCFG 中的 DBGFORCE 寄存器可用于将 SW-DP 与外部调试引脚分离，转而由软件直接比特操作内部 SWD 信号。该功能旨在让一个核心上运行的调试探针用于调试另一核心。对于其他使用场景，通常更为妥当的做法是使用自托管调试访问，直接从系统总线与 AP 进行接口。

### 3.5.6. 自托管调试

图 10 中所示的所有 AP（除 RP-AP 外）均支持通过系统总线的直接内存映射访问。这称为自托管调试，因为经妥善操作，它允许在系统内直接运行调试主机（即调试器）。该功能还可用于访问跟踪硬件，从而借助跟踪 DMA FIFO 实现自托管跟踪。默认仅允许安全访问，因为处理器调试为非安全代码提供了干扰安全上下文及/或执行安全总线访问的可能。

自托管调试窗口起始地址为 `0x40140000` (CORESIGHT\_PERIPH\_BASE)。该窗口内的 AP 偏移量与从 SW-DP 访问 AP 时的地址相同。

由于 AHB-AP 的 DRW 寄存器具有阻塞特性，且其与 Cortex-M33 对 AHB-AP 访问的仲裁机制以及 load/store 操作相互作用，部分访问可能引发因总线循环阻塞依赖而导致的总线死锁。特别地，核心不得通过自托管调试窗口访问其自身的 AHB-AP，且 AHB-AP 不得通过自托管调试窗口访问其他 AHB-AP——尝试这样做将立即返回总线故障。为减少死锁风险，采用完整的 APB 交叉开关，将 SW-DP 和自托管调试端口连接至 AP；例如，自托管使用 Arm 跟踪硬件时，不会干扰通过 AHB-AP 连接的外部调试器。

在某些情况下，无法避免总线死锁，例如一个核心通过自托管调试窗口，使用另一个核心的 AHB-AP 访问其他 APB 外设时：

1. APB 的 DRW 寄存器上游访问在下游访问完成之前不会完成。
2. 下游访问在获得系统 APB 桥访问权限之前不会完成。
3. 访问 APB 桥的权限在占用系统 APB 桥的上游访问完成之前不会被授予。
4. 参见第 1 点。

当在一个核心上运行自托管调试器，调试访问 APB 地址的另一个核心上的代码时，可能会出现这种情况。当 APB 桥的 65536 周期超时到期，放弃传输并向上游访问发起方返回总线错误后，死锁最终被解除。为避免此情况，软件应在即将使用 AP 访问 APB 地址（以 `0x4` 开头的地址）时进行检测，并直接执行访问，而非使用 Mem-AP。

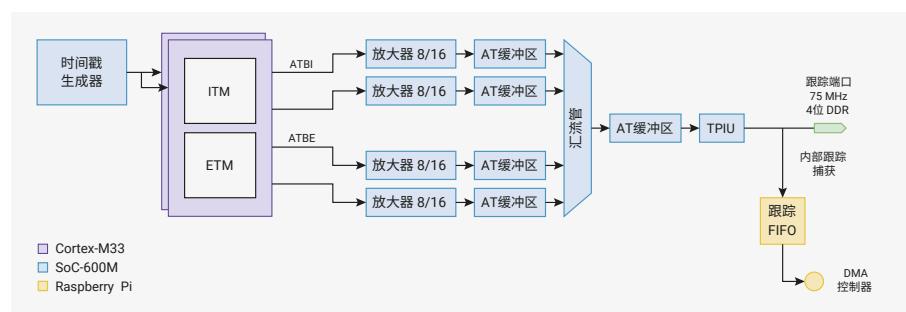
当调试器通过 RISC-V 系统总线访问总线时，不会发生此类死锁，因为 DM 上游的总线传输不会因下游访问完成而阻塞。

### 3.5.7. 跟踪

#### 3.5.7.1 概述

ATB 跟踪子系统基于 Coresight SoC-600M 架构，如图 11 所示。

图 11 跟踪子系统



跟踪子系统捕获来自每个 Cortex-M33 ITM/ETM 组件的跟踪消息，将其合并为单一跟踪总线，并通过 4 位 DDR 跟踪端口发送至片外，供跟踪端口分析仪进行后续捕获和分析。

这使开发人员能够审查处理器上执行的软件的详细日志。与传统硬件调试相比，其优点在于无需暂停处理器或影响其执行时序，从而能够诊断在调试器下难以重现的软件问题。

跟踪子系统包含以下主要组件：

- 时间戳生成器：时间戳传播至两个 Cortex-M33 处理器，并且应用于 ETM 和 ITM 输出，以便恢复其跟踪流的相对时序。
- Cortex-M33 ETM：嵌入式跟踪宏单元，用于基于对 Cortex-M33 执行过程观察生成的实时指令流消息。
- Cortex-M33 ITM：指令跟踪宏单元，用于软件生成的消息。
- ATB 汇聚器：利用时间戳生成器的时间戳，将 Cortex-M33 的跟踪源合并为单一跟踪流。
- TPIU：跟踪端口接口单元，通过跟踪端口引脚输出跟踪数据。源同步跟踪接口为 4 位 DDR，最高时钟频率 75 MHz，最大跟踪数据速率可达 600 Mb/s。

- 跟踪FIFO：可选地在设备上捕获32位TPIU跟踪流，从此处开始，DMA可将数据传输至主系统SRAM。

有关Cortex-M33 ETM的详细信息，请参见Arm CoreSight ETM-M33技术参考手册。有关图11中其他跟踪组件的详细信息，请参见SoC-600M技术参考手册。

跟踪输出时钟固定为 `clk_sys` 的一半。在系统最高频率150 MHz时，TPIU输出时钟为75 MHz。在较低系统时钟频率时，跟踪吞吐量会降低，但实际应用中很少成为问题，因为处理器指令吞吐量（及对跟踪输出带宽的需求）会相应调整。

### 3.5.7.2. 跟踪FIFO

跟踪输出传输至两个数据接收端之一：

- 四位TPIU接口通过GPIO将数据从芯片中发送出去，以供外部探针捕获。
- 跟踪FIFO通过系统DMA将数据流入SRAM。

DMA的带宽高于TPIU接口的带宽。捕获至片上缓冲区亦允许跟踪通过相对低速的SWD探针运行，且不会限制跟踪带宽。

其操作方式类似于微追踪缓冲器（MTB）。然而，系统SRAM全部可用于跟踪。您亦可利用其他DMA端点，如PIO和HST X，实现自定义跟踪数据接收端，例如当您希望使用比TPIU提供更宽且频率更低的总线时。

在尝试使用DMA从该FIFO读取数据前，必须通过设置ACCESSCTRL CORESIGHT\_TRACE寄存器中的 `DMA` 位来启用对跟踪FIFO寄存器的DMA访问。配置DMA的DREQ为 [53](#) 以选择跟踪FIFO。

### 3.5.7.3. 跟踪FIFO寄存器列表

跟踪FIFO寄存器的基地址起始于 [0x50700000](#)（在SDK中定义为CORESIGHT\_TRACE\_BASE）。

表96。CORESIGHT\_TRACE寄存器列表

| 偏移量 | 名称                                 | 说明                 |
|-----|------------------------------------|--------------------|
| 0x0 | <a href="#">CTRL_STATUS</a>        | 控制与状态寄存器           |
| 0x4 | <a href="#">TRACE_CAPTURE_FIFO</a> | 用于从TPIU采集跟踪数据的FIFO |

## CORESIGHT\_TRACE: CTRL\_STATUS寄存器

偏移：0x0

### 描述

控制与状态寄存器

表97。CTRL\_STATUS寄存器

| 位    | 描述                                                                           | 类型 | 复位  |
|------|------------------------------------------------------------------------------|----|-----|
| 31:2 | 保留。                                                                          | -  | -   |
| 1    | <b>TRACE_CAPTURE_FIFO_OVERFLOW</b> ：当FIFO已满导致采样的跟踪数据丢失时，此状态标志置位。写入1以确认并清除此位。 | 读写 | 0x0 |

| 位 | 描述                                                                                                                                                                                                                        | 类型 | 复位  |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 0 | <p><b>TRACE_CAPTURE_FIFO_FLUSH:</b> 写1以持续将跟踪FIFO保持在刷新状态，防止溢出。</p> <p>清除此标志之前，请配置并启动正确DREQ的DMA通道，用于TRACE_CAPTURE_FIFO寄存器。</p> <p>清除此标志以开始采样跟踪数据，跟踪捕获缓冲区满时该标志再次置位。您必须配置TPIU以生成可捕获的跟踪数据包，并且需要配置如ETM等上游组件，以生成传递至TPIU的事件流。</p> | 读写 | 0x1 |

## CORESIGHT\_TRACE: TRACE\_CAPTURE\_FIFO寄存器

偏移量: 0x4

### 描述

用于从TPIU采集跟踪数据的FIFO

表98。  
TRACE\_CAPTURE\_FIFO  
0寄存器

| 位    | 描述                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 类型 | 复位         |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | <p><b>RDATA:</b> 从包含由TPIU捕获的跟踪数据的8×32位FIFO中读取。</p> <p>硬件在clk_sys上升沿将数据推送至FIFO，满足以下任一条件时生效：</p> <ul style="list-style-type: none"> <li>* TPIU TRACETCL输出为低电平（正常跟踪数据）</li> <li>* TPIU TRACETCL输出为高电平，且TPIU TRACEDATA0和TRACEDATA1均为低电平（触发包）</li> </ul> <p>上述条件符合Arm Coresight Architecture Spec v3.0第D3.3.3节：“跟踪捕获设备解码要求”</p> <p>被捕获至FIFO的数据为TPIU输出的完整32位TRACEDATA总线。请注意，TPIU采用DDR输出，频率为clk_sys的一半，因此该接口通过在TPIU输出时钟每个有效边缘采样，能够捕获完整32位TPIU DDR输出带宽。</p> | RF | 0x00000000 |

### 3.5.8. 救援重置

救援复位是一种完整的系统复位，类似于使RUN引脚拉低，同时设置一个标志，指示只读存储器在执行任何用户软件之前停止运行。该操作通过RP-AP在SWD总线上完成，即使系统时钟停止且切换核心电源域断电时也能执行。当芯片锁死时使用此功能，例如代码已被编程进闪存，导致系统时钟永久停止：

由于调试器无法再与处理器通信以恢复系统正常状态，因此需要采取更为严厉的措施。此功能由RP2040上的Rescue DP提供，而在RP2350上则由RP-AP提供，以避免强制使用多点SWD。

通过在RP-AP中设置并随后清除CTRL.RESCUE\_RESTART位来调用救援功能。此操作将导致芯片硬复位，并设置CHIP\_RESET.RESCUE\_FLAG，表明已执行救援复位。只读存储器引导程序在初始启动过程中几乎立即检查此标志（位于看门狗、闪存或USB启动之前），通过清除此位确认，然后停止处理器。这使系统保持在安全状态，系统时钟继续运行，以便调试器能够重新连接核心并加载新的代码。

### 3.5.9. 安全

默认情况下，SWD调试访问端口允许外部调试器访问全部系统内存和外设，观察并修改处理器上执行的软件。如果启用了引导签名强制（参见第10.1.1节），调试访问将成为安全隐患，因为它能够绕过该保护。为此，RP2350通过片上OTP存储中的配置支持逐步锁定调试端口。

概念上存在两个控制位：调试禁用和安全调试禁用。调试禁用旨在完全切断对处理器及系统总线的调试访问，而安全调试禁用禁止安全总线访问及在安全状态下暂停处理器，但仍允许对非安全软件进行正常调试。设置这些控制位有两种方式：

- 设置相关的OTP关键标志：CRIT1.DEBUG\_DISABLE或CRIT1.SECURE\_DEBUG\_DISABLE，分别用于配置调试禁用或安全调试禁用。
- 将128位固定调试密钥安装为OTP密钥5或6（详见第3.5.9.2节）。

OTP配置的变更将在下一次OTP块复位时生效。

一旦调试被禁用，软件可以使用OTP DEBUGEN寄存器重新启用调试，该寄存器允许针对每个处理器单独清除安全调试和整体调试启用标志。例如，安全软件可实现一个shell，用户通过加密挑战进行身份验证，从而在默认禁用调试的系统上启用调试功能。DEBUGEN寄存器属于处理器冷复位域，因此从OTP（第二PSM阶段）开始，它可在PSM复位期间保持不变。这允许几乎完整的系统复位而不丢失调试访问权限。

为防止DEBUGEN寄存器发生意外写入，其各位可通过DEBUGEN\_LOCK寄存器中的对应位单独加锁。

本产品提供以下逐级增强的调试保护级别：

1. 完全开放：未安装任何密钥，且未设置任何OTP调试禁用标志。此配置最适合产品开发阶段。
2. 仅限持密钥访问：已安装至少一个密钥，但未设置任何OTP调试禁用标志。
3. 即使持有密钥也无法访问（已设置OTP调试禁用标志），但安全代码可通过写入DEBUGEN启用调试访问。
4. 即使持有密钥也无法访问（已设置OTP调试禁用标志），且DEBUGEN已被DEBUGEN\_LOCK锁定。

#### 3.5.9.1. 调试禁用的影响

安全调试禁用标志（CRIT1.SECURE\_DEBUG\_DISABLE）具有以下效用：

- 将Arm核0和核1的AHB-AP安全AP启用信号设为0。
  - 此操作阻止AP进行安全总线访问（包括访问PPB）。
  - 状态通过AHB-AP的CSW寄存器中的SDeviceEn标志反映。
- 将两个核的Cortex-M33 SPIDEN和SPNIDEN信号设为0。
  - 这会防止内核在安全状态下被暂停或跟踪。
- 禁用工厂测试JTAG接口（第10.10节）。

### 注意

两个AHB-AP的`CSW.HNONSEC`位默认均为0，产生安全总线访问。若设置了安全调试禁用标志，则这些位必须设置为1，以产生非安全总线访问。

调试禁用标志（CRIT1.DEBUG\_DISABLE）具备与安全调试禁用标志相同的所有效应。其还具有以下附加效应：

- 将Arm核心0和核心1的AHB-AP使能信号设为0。
  - 这将阻止AP执行任何总线访问（包括对PPB的访问）。
  - 状态通过AHB-AP `CSW`寄存器中的 `DeviceEn`标志进行报告。
- 将RISC-V DM APB-AP的AP使能信号设为0。
  - 这将阻止AP访问RISC-V调试模块。
  - 状态由APB-AP `CSW`寄存器中的 `DeviceEn`标志报告。
- 将CTI的 `DBGEN`和 `NIDEN`信号置为0。

在RISC-V中，CRIT1.SECURE\_DEBUG\_DISABLE无实际效果。核心的调试模式访问始终具有安全且特权的总线属性，除非被FORCE\_CORE\_NS降低。同样，系统总线通过调试模块的访问始终为安全且特权，除非设置了FORCE\_CORE\_NS.CORE1，此时为非安全且特权。请使用RISC-V上的CRIT1.DEBUG\_DISABLE标志。

#### 3.5.9.2. 调试密钥

第13.5.2节描述了OTP硬件访问密钥。硬件在OTP复位后的OTP上电序列中，将OTP访问密钥加载到隐藏寄存器，随后相应的OTP位置变为不可访问。OTP密钥 5和 6具有特殊作用，除作为普通OTP页密钥外，还控制对SWD调试硬件的访问权限。

调试密钥为一个128位固定挑战值。在OTP中安装调试密钥将锁定调试访问，且该状态将持续，直至调试主机通过RP-AP `DBGKEY`寄存器写入匹配密钥值。此接口为只写。

要安装调试密钥，需先从KEY5\_0或KEY6\_0开始编程OTP位置，这些位置受ECC保护。编程完128位密钥值并读取确认无误后，写入原始位模式 `0x010101`至KEY5\_VALID或KEY6\_VALID，以标记密钥为有效。该有效性将在OTP块的下一次复位时生效。

一旦密钥生效，该密钥对应的OTP存储位置将无法再被读写。仅OTP上电状态机（第13.3.4节）可读取该密钥。

调试密钥安装的作用取决于已安装的密钥 5和/或 6：

- 如果密钥5或密钥6有效，且通过RP-AP未输入任何匹配密钥，则所有调试功能均被禁用。  
此操作等同于设置CRIT1.DEBUG\_DISABLE。
- 如果密钥5有效，且通过RP-AP未输入匹配密钥（特别是密钥5），则安全调试被禁用。此操作等同于写入CRIT1.SECURE\_DEBUG\_DISABLE。

当同时安装密钥时，密钥 5同时提供安全和非安全调试访问；密钥 6仅提供非安全调试访问。当仅安装单个密钥时，该密钥同时提供安全和非安全调试访问。

通过SWD输入密钥时，首先向`DBGKEY.RESET`写入 1。然后依次向`DBGKEY.DATA`写入128位数据，每写入一位，同时向`DBGKEY.PUSH`写入 1。数据应以LSB优先顺序写入，起始于编号最低的OTP行。

假设您写入的值匹配已安装的某个调试密钥，则在第128次按压后，调试功能将被解锁。Mem-APCSW寄存器中的`SDeviceEn`和`DeviceEn`标志指示操作的成功或失败。

若未通过RP-AP提供匹配的密钥，则调试功能将被禁用（若原本应启用）。然而，若因其他原因已被禁用，则即使提供密钥，调试功能也不会启用。例如，当CRIT1.DEBUG\_DISABLE被设置，且

DEBUGEN位被清除时，无论调试密钥和RP-AP的状态如何，调试功能均被禁用。

### 3.5.10. RP-AP

RP-AP为一个小型寄存器块，始终可通过SWD访问。访问RP-AP无需切换核心域供电，也无需任何内部系统时钟发生器运行。

#### 3.5.10.1. 寄存器列表

RP-AP寄存器起始于调试地址空间偏移量 `0x80000`，访问地址位于SW-DP的SELECT寄存器中的 `0x80000`。与其他AP不同，无法直接通过系统总线访问。

表99.  
RP\_AP寄存器列表

| 偏移量   | 名称                                   | 说明                                                                                                                                                                                                                                                                                                                                             |
|-------|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x000 | <a href="#">CTRL</a>                 | 该寄存器主要用于DFT，也可用于解决部分上电问题。但不应使用该寄存器强制域的上电。请使用DBG_POW_OVRD实现此功能。                                                                                                                                                                                                                                                                                 |
| 0x004 | <a href="#">DBGKEY</a>               | 串行密钥加载接口（仅写）                                                                                                                                                                                                                                                                                                                                   |
| 0x008 | <a href="#">DBG_POW_STATE_SWCORE</a> | 该寄存器指示切换核心域电源序列器的状态。<br><br>序列器的时序由POWMAN_SEQ_*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。<br><br>域上电通过清除第0位（IS_PD）开始。<br>然后依次设置第1至第8位。第8位（IS_PU）指示序列完成。<br><br>域断电通过清除第8位（IS_PU）开始，然后依次清除第7至第1位。<br>随后将位0（IS_PU）设置，以指示序列已完成。<br><br>位9至11描述电源管理时钟的状态，该状态随着切换核心电源启动后切换核心中时钟发生器的可用性而变化。<br><br>该总线可发送至GPIO用于调试。详见<br>DBG_POW_OVRD寄存器中的DBG_POW_OUTPUT_TO_GPIO。 |
| 0x00c | <a href="#">DBG_POW_STATE_XIP</a>    | 此寄存器指示XIP域电源顺序器的状态。<br><br>序列器的时序由POWMAN_SEQ_*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。<br><br>域上电通过清除第0位（IS_PD）开始。<br>然后依次设置第1至第8位。第8位（IS_PU）指示序列完成。<br><br>域断电通过清除第8位（IS_PU）开始，然后依次清除第7至第1位。<br>随后将位0（IS_PU）设置，以指示序列已完成。                                                                                                                                 |

| 偏移量   | 名称                     | 说明                                                                                                                                                                                                                                                                                                                              |
|-------|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x010 | DBG_POW_STATE_SRAM0    | <p>此寄存器指示SRAM0域电源顺序器的状态。</p> <p>序列器的时序由POWMAN_SEQ_*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。</p> <p>域上电通过清除第0位 (IS_PD) 开始。<br/>然后依次设置第1至第8位。第8位 (IS_PU) 指示序列完成。</p> <p>域断电通过清除第8位 (IS_PU) 开始，然后依次清除第7至第1位。<br/>随后将位0 (IS_PU) 设置，以指示序列已完成。</p>                                                                                               |
| 0x014 | DBG_POW_STATE_SRAM1    | <p>此寄存器指示SRAM1域电源顺序器的状态。</p> <p>序列器的时序由POWMAN_SEQ_*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。</p> <p>域上电通过清除第0位 (IS_PD) 开始。<br/>然后依次设置第1至第8位。第8位 (IS_PU) 指示序列完成。</p> <p>域断电通过清除第8位 (IS_PU) 开始，然后依次清除第7至第1位。<br/>随后将位0 (IS_PU) 设置，以指示序列已完成。</p>                                                                                               |
| 0x018 | DBG_POW_OVRD           | <p>该寄存器允许对所有切换电源域的电源顺序器输出进行外部控制。如果任何电源顺序器在任一阶段停滞，则通过执行以下序列强制所有域的电源开启：</p> <ul style="list-style-type: none"> <li>- 将DBG_POW_OVRD设置为0x3b，以强制小电源开关开启、大电源开关关闭、复位开启以及隔离开启</li> <li>- 允许域电源达到全轨电压</li> <li>- 设置DBG_POW_OVRD = 0x3b以强制开启大型电源开关</li> <li>- 设置DBG_POW_OVRD = 0x37以解除隔离</li> <li>- 设置DBG_POW_OVRD = 0x17以解除复位</li> </ul> |
| 0x01c | DBG_POW_OUTPUT_TO_GPIO | <p>将DBG_POW_STATE_SWCORE的部分或全部位发送至GPIO。</p> <p>位0将DBG_POW_STATE_SWCORE的第0位发送至GPIO 34<br/>位1将DBG_POW_STATE_SWCORE的第1位发送至GPIO 35<br/>位2将DBG_POW_STATE_SWCORE的第2位发送至GPIO 36<br/>.<br/>. .<br/>位11将DBG_POW_STATE_SWCORE的第11位发送至GPIO 45</p>                                                                                          |
| 0xdfc | IDR                    | 标准Coresight ID寄存器                                                                                                                                                                                                                                                                                                               |

## RP\_AP: CTRL 寄存器

偏移：0x000

### 描述

该寄存器主要用于DFT，也可用于解决部分上电问题。但不应使用该寄存器强制域的上电。请使用DBG\_POW\_OVRD实现此功能。

表 100. CTRL  
寄存器

| 位    | 描述                                                                                                                                                                                                                                                 | 类型 | 复位  |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31   | <b>RESCUE_RESTART</b> : 通过以最小引导代码执行重启芯片，允许对引导问题进行调试。写入1以将芯片置于复位状态，随后写入0，在设定救援标志后重新启动芯片。救援标志位于POWMAN_CHIP_RESET寄存器，由引导程序读取。通过向POWMAN_CHIP_RESET_RESCUE_FLAG写入0，或通过除RESCUE_RESTART以外的任何方式复位芯片，可清除救援标志。                                               | 读写 | 0x0 |
| 30   | <b>SPARE</b> : 未使用                                                                                                                                                                                                                                 | 读写 | 0x0 |
| 29:7 | 保留。                                                                                                                                                                                                                                                | -  | -   |
| 6    | <b>DBG_FRCE_GPIO_LPCK</b> : 当低功耗振荡器 (LPOSOC) 失效或异常时，允许芯片启动，并允许调整初始电源序列速率。写入1以强制将LPOSOC输出驱动切换至GPIO（80针封装为gpio20，60针封装为gpio34）。<br><br>若LPOSOC失效或异常，可能还需设置此寄存器中的LPOSOC_STABLE_FRCE位。用户必须在GPIO上提供时钟信号。正常运行时，时钟频率应约为32kHz。<br><br>调整频率将加快或减慢初始上电序列的速度。 | 读写 | 0x0 |
| 5    | <b>LPOSOC_STABLE_FRCE</b> : 允许芯片启动，即使低功耗振荡器 (LPOSOC) 未能设置其稳定标志。初始上电时序由约32kHz的LPOSOC提供时钟，但直到LPOSOC声明稳定前，此时序不会启动。如果LPOSOC正常工作，设置此位后芯片将启动；若LPOSOC异常，则必须设置DBG_FRCE_GPIO_LPCK并提供外部时钟。                                                                   | 读写 | 0x0 |
| 4    | <b>POWMAN_DFT_ISO_OFF</b> : 保持电源域之间的隔离门处于打开状态。此功能用于为DFT和电源管理调试保持隔离门开启。该功能并非用于强制隔离门开启。可使用DBG_POW_OVRD中的覆盖功能强制隔离门开启或关闭。                                                                                                                              | 读写 | 0x0 |
| 3    | <b>POWMAN_DFT_PWRON</b> : 保持所有电源域的电源开关处于开启状态。此功能旨在保持电源供给以支持DFT和调试，而非用于开启电源。电源开关未按顺序切换，突发的电流需求可能导致始终通电域电压下降。该寄存器位于始终通电域，故可能引发混乱。建议使用DBG_POW_OVRD控制。                                                                                                 | 读写 | 0x0 |
| 2    | <b>POWMAN_DBGMODE</b> : 防止电源管理器关闭电源及重置切换核心电源域。设计用于芯片启动后对电源管理器进行DFT及调试。不可用于强制初始通电，因为该操作会同时解除复位信号。                                                                                                                                                   | 读写 | 0x0 |
| 1    | <b>JTAG_FUNCSEL</b> : 将JTAG端口复用至GPIO0-3。                                                                                                                                                                                                           | 读写 | 0x0 |
| 0    | <b>JTAG_TRSTN</b> : 复位JTAG模块，低电平有效。                                                                                                                                                                                                                | 读写 | 0x0 |

## RP\_AP: DBGKEY寄存器

偏移量: 0x004

### 描述

串行密钥加载接口（仅写）

表101. DBGKEY寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:3 | 保留。 | -  | -  |

| 位 | 描述                           | 类型 | 复位  |
|---|------------------------------|----|-----|
| 2 | <b>RESET</b> : 复位 (发送新密钥前使用) | 读写 | 0x0 |
| 1 | <b>PUSH</b>                  | 读写 | 0x0 |
| 0 | <b>DATA</b>                  | 读写 | 0x0 |

## RP\_AP: DBG\_POW\_STATE\_SWCORE寄存器

偏移: 0x008

### 描述

该寄存器指示切换核心域电源序列器的状态。

序列器的时序由POWMAN\_SEQ\_\*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。

域的上电通过清除第0位 (IS\_PD) , 随后依次设置第1至第8位开始。第8位 (IS\_PU) 指示序列完成。

域断电通过清除第8位 (IS\_PU) 开始, 然后依次清除第7至第1位。随后将位0 (IS\_PU) 设置, 以指示序列已完成。

位9至11描述电源管理时钟的状态, 该状态随着切换核心电源启动后切换核心中时钟发生器的可用性而变化。

该总线可通过GPIO进行调试。详见DBG\_POW\_OVRD寄存器中的DBG\_POW\_OUTPUT\_TO\_GPIO。

表102。  
DBG\_POW\_STATE\_SW  
CORE寄存器

| 位     | 描述                                                                                                                                                                                                                                                                                                                               | 类型 | 复位  |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:12 | 保留。                                                                                                                                                                                                                                                                                                                              | -  | -   |
| 11    | <b>USING_FAST_POWCK</b> : 指示电源管理时钟的来源。在切换核心上电时, 时钟由LPOSOC切换至clk_ref, 该标志置位。clk_ref最初由ROSC提供, 但在XOSC可用后切换至XOSC。在切换核心断电时, 时钟切换至LPOSOC, 该标志被清除。                                                                                                                                                                                     | 只读 | 0x0 |
| 10    | <b>WAITING_POWCK</b> : 指示切换核心电源序列器正在等待电源管理时钟更新。切换核心上电时, 时钟由LPOSOC切换至clk_ref。clk_ref初始由ROSC驱动运行, 但当XOSC可用时将切换至XOSC。切换核心断电时, 时钟切换至LPOSOC。<br><br>若切换核心上电序列在该标志激活时停止, 表示clk_ref未运行, 说明ROSC存在问题。发生此情况时, 应在DBG_POW_OVRD寄存器中设置DBG_POW_RSTSTART_FROM_XOSC, 以避免使用ROSC。<br><br>若切换核心断电序列在该标志激活时停止, 表示LPOSOC未运行。解决方案是在切换核心电源域供电时不停止LPOSOC。 | 只读 | 0x0 |
| 9     | <b>WAITING_TIMCK</b> : 指示切换核心电源顺序器正在等待AON计时器更新。切换核心上电时无需执行任何操作。AON计时器继续由LPOSOC驱动运行, 因此该标志不会被设置。软件决定是否将AON计时器时钟切换至XOSC (通过clk_ref)。在切换核心断电时, 序列器将把AON计时器切换回LPOSOC, 若软件已将其切换至XOSC。切换过程中, WAITING_TIMCK标志将被置位。如切换核心断电序列在此标志激活时停滞, 唯一方案是重置芯片, 并调整软件以避免选择XOSC作为AON计时器时钟源。                                                             | 只读 | 0x0 |
| 8     | <b>IS_PU</b> : 表示电源域已完全上电。                                                                                                                                                                                                                                                                                                       | 只读 | 0x0 |
| 7     | <b>RESET_FROM_SEQ</b> : 表示对电源域的复位状态。                                                                                                                                                                                                                                                                                             | 只读 | 0x0 |

| 位 | 描述                                                                                                                | 类型 | 复位  |
|---|-------------------------------------------------------------------------------------------------------------------|----|-----|
| 6 | <b>ENAB_ACK</b> : 表示对电源域的使能状态。                                                                                    | 只读 | 0x0 |
| 5 | <b>ISOLATE_FROM_SEQ</b> : 表示对电源域的隔离控制状态。                                                                          | 只读 | 0x0 |
| 4 | <b>LARGE_ACK</b> : 表示电源域大功率开关的状态。                                                                                 | 只读 | 0x0 |
| 3 | <b>SMALL_ACK2</b> : 小功率开关分为3条链。在上电序列中，这些开关被分别打开，以便控制VDD的上升时间。在断电序列中，它们与大型电源开关同时关闭。<br><br>该位指示小型电源开关链2中最后一个元件的状态。 | 只读 | 0x0 |
| 2 | <b>SMALL_ACK1</b> : 该位指示小型电源开关链1中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 1 | <b>SMALL_ACK0</b> : 该位指示小型电源开关链0中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 0 | <b>IS_PD</b> : 指示电源域已完全断电。                                                                                        | 只读 | 0x0 |

## RP\_AP: DBG\_POW\_STATE\_XIP 寄存器

偏移量: 0x00c

### 描述

此寄存器指示XIP域电源顺序器的状态。

序列器的时序由POWMAN\_SEQ\_\*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。

域的上电通过清除第0位 (IS\_PD) ，随后依次设置第1至第8位开始。第8位 (IS\_PU) 指示序列完成。

域断电通过清除第8位 (IS\_PU) 开始，然后依次清除第7至第1位。随后将位0 (IS\_PU) 设置，以指示序列已完成。

表103。  
DBG\_POW\_STATE\_XIP  
寄存器

| 位    | 描述                                                                                                                | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:9 | 保留。                                                                                                               | -  | -   |
| 8    | <b>IS_PU</b> : 表示电源域已完全上电。                                                                                        | 只读 | 0x0 |
| 7    | <b>RESET_FROM_SEQ</b> : 表示对电源域的复位状态。                                                                              | 只读 | 0x0 |
| 6    | <b>ENAB_ACK</b> : 表示对电源域的使能状态。                                                                                    | 只读 | 0x0 |
| 5    | <b>ISOLATE_FROM_SEQ</b> : 表示对电源域的隔离控制状态。                                                                          | 只读 | 0x0 |
| 4    | <b>LARGE_ACK</b> : 表示电源域大功率开关的状态。                                                                                 | 只读 | 0x0 |
| 3    | <b>SMALL_ACK2</b> : 小功率开关分为3条链。在上电序列中，这些开关被分别打开，以便控制VDD的上升时间。在断电序列中，它们与大型电源开关同时关闭。<br><br>该位指示小型电源开关链2中最后一个元件的状态。 | 只读 | 0x0 |
| 2    | <b>SMALL_ACK1</b> : 该位指示小型电源开关链1中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 1    | <b>SMALL_ACK0</b> : 该位指示小型电源开关链0中最后一个元件的状态。                                                                       | 只读 | 0x0 |

| 位 | 描述                         | 类型 | 复位  |
|---|----------------------------|----|-----|
| 0 | <b>IS_PD</b> : 指示电源域已完全断电。 | 只读 | 0x0 |

## RP\_AP: DBG\_POW\_STATE\_SRAM0 寄存器

偏移量: 0x010

### 描述

此寄存器指示SRAM0域电源顺序器的状态。

序列器的时序由POWMAN\_SEQ\_\*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。

域的上电通过清除第0位 (IS\_PD) ，随后依次设置第1至第8位开始。第8位 (IS\_PU) 指示序列完成。

域断电通过清除第8位 (IS\_PU) 开始，然后依次清除第7至第1位。随后将位0 (IS\_PU) 设置，以指示序列已完成。

表104。  
DBG\_POW\_STATE\_SRAM0 寄存器

| 位    | 描述                                                                                                                | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:9 | 保留。                                                                                                               | -  | -   |
| 8    | <b>IS_PU</b> : 表示电源域已完全上电。                                                                                        | 只读 | 0x0 |
| 7    | <b>RESET_FROM_SEQ</b> : 表示对电源域的复位状态。                                                                              | 只读 | 0x0 |
| 6    | <b>ENAB_ACK</b> : 表示对电源域的使能状态。                                                                                    | 只读 | 0x0 |
| 5    | <b>ISOLATE_FROM_SEQ</b> : 表示对电源域的隔离控制状态。                                                                          | 只读 | 0x0 |
| 4    | <b>LARGE_ACK</b> : 表示电源域大功率开关的状态。                                                                                 | 只读 | 0x0 |
| 3    | <b>SMALL_ACK2</b> : 小功率开关分为3条链。在上电序列中，这些开关被分别打开，以便控制VDD的上升时间。在断电序列中，它们与大型电源开关同时关闭。<br><br>该位指示小型电源开关链2中最后一个元件的状态。 | 只读 | 0x0 |
| 2    | <b>SMALL_ACK1</b> : 该位指示小型电源开关链1中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 1    | <b>SMALL_ACK0</b> : 该位指示小型电源开关链0中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 0    | <b>IS_PD</b> : 指示电源域已完全断电。                                                                                        | 只读 | 0x0 |

## RP\_AP: DBG\_POW\_STATE\_SRAM1 寄存器

偏移量: 0x014

### 描述

此寄存器指示SRAM1域电源顺序器的状态。

序列器的时序由POWMAN\_SEQ\_\*寄存器控制。有关时序的更多信息，请参阅这些寄存器的头文件。

域的上电通过清除第0位 (IS\_PD) ，随后依次设置第1至第8位开始。第8位 (IS\_PU) 指示序列完成。

域断电通过清除第8位 (IS\_PU) 开始，然后依次清除第7至第1位。随后将位0 (IS\_PU) 设置，以指示序列已完成。

表105。  
DBG\_POW\_STATE\_S  
RAM1寄存器

| 位    | 描述                                                                                                                | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:9 | 保留。                                                                                                               | -  | -   |
| 8    | <b>IS_PU</b> : 表示电源域已完全上电。                                                                                        | 只读 | 0x0 |
| 7    | <b>RESET_FROM_SEQ</b> : 表示对电源域的复位状态。                                                                              | 只读 | 0x0 |
| 6    | <b>ENAB_ACK</b> : 表示对电源域的使能状态。                                                                                    | 只读 | 0x0 |
| 5    | <b>ISOLATE_FROM_SEQ</b> : 表示对电源域的隔离控制状态。                                                                          | 只读 | 0x0 |
| 4    | <b>LARGE_ACK</b> : 表示电源域大功率开关的状态。                                                                                 | 只读 | 0x0 |
| 3    | <b>SMALL_ACK2</b> : 小功率开关分为3条链。在上电序列中，这些开关被分别打开，以便控制VDD的上升时间。在断电序列中，它们与大型电源开关同时关闭。<br><br>该位指示小型电源开关链2中最后一个元件的状态。 | 只读 | 0x0 |
| 2    | <b>SMALL_ACK1</b> : 该位指示小型电源开关链1中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 1    | <b>SMALL_ACK0</b> : 该位指示小型电源开关链0中最后一个元件的状态。                                                                       | 只读 | 0x0 |
| 0    | <b>IS_PD</b> : 指示电源域已完全断电。                                                                                        | 只读 | 0x0 |

## RP\_AP: DBG\_POW\_OVRD 寄存器

偏移量: 0x018

### 描述

该寄存器允许对所有切换电源域的电源顺序器输出进行外部控制。如果任何电源顺序器在任一阶段停滞，则通过执行以下序列强制所有域的电源开启：

- 设置 **DBG\_POW\_OVRD** = 0x3b，强制小功率开关闭合，大功率开关断开，复位开启，并启用隔离
- 等待电源域电源达到满电压
- 设置 **DBG\_POW\_OVRD** = 0x3b，强制大功率开关闭合
- 设置 **DBG\_POW\_OVRD** = 0x37，取消隔离
- 设置 **DBG\_POW\_OVRD** = 0x17，取消复位

表 106。  
DBG\_POW\_OVRD  
寄存器

| 位    | 描述                                                                                                                                                                                                                                                                                                                              | 类型 | 复位  |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:7 | 保留。                                                                                                                                                                                                                                                                                                                             | -  | -   |
| 6    | <b>DBG_POW_RESTART_FROM_XOSC</b> : 系统默认在 ROSC 提供时钟后立即启动，待 XOSC 可用时切换至 XOSC。之所以如此，是因为 XOSC 启动需数毫秒。如 ROSC 出现问题，可更改默认行为，停止使用 ROSC，改为等待 XOSC。但这需修改电源管理器 START_FROM_XOSC 寄存器的复位值掩码。<br><br>为了允许实验，默认值可以通过将此寄存器位设置为1来临时更改。设置此位后，必须通过Coresight dprst或救援复位（参见上述RP_AP_CTRL寄存器中的RESCUE_RESTART）对内核进行复位。上电复位、欠压复位或RUN引脚复位将复位该控制位，并恢复为默认行为。 | 读写 | 0x0 |

| 位 | 描述                                                                                                        | 类型 | 复位  |
|---|-----------------------------------------------------------------------------------------------------------|----|-----|
| 5 | <b>DBG_POW_RESET:</b> 当DBG_POW_OVRD_RESET=1时，此寄存器位控制所有域的复位。1=复位，0=不复位。                                    | 读写 | 0x0 |
| 4 | <b>DBG_POW_OVRD_RESET:</b> 使DBG_POW_RESET能够控制电源管理器和切换内核的复位。基本上，除了Coresight双线接口和RP_AP寄存器外，涵盖所有内容。          | 读写 | 0x0 |
| 3 | <b>DBG_POW_ISO:</b> 当DBG_POW_OVRD_ISO=1时，此寄存器位控制所有域的隔离闸门。1=隔离，0=未隔离。                                      | 读写 | 0x0 |
| 2 | <b>DBG_POW_OVRD_ISO:</b> 使能DBG_POW_ISO以控制域间隔离门。                                                           | 读写 | 0x0 |
| 1 | <b>DBG_POW_OVRD_LARGE_REQ:</b> 开启所有域的大功率开关。应在小功率开关充分升压后进行此操作。过早开启大功率开关会导致始终开启域电压降低，进而损坏相关寄存器。             | 读写 | 0x0 |
| 0 | <b>DBG_POW_OVRD_SMALL_REQ:</b> 开启所有域的小功率开关。此操作开启每个域的链路0，同时关闭链路2、链路3及大功率开关链路。该操作将提升所有域的电源，且不会使始终开启电源域电压下降。 | 读写 | 0x0 |

## RP\_AP: DBG\_POW\_OUTPUT\_TO\_GPIO 寄存器

偏移量: 0x01c

### 描述

将DBG\_POW\_STATE\_SWCORE的部分或全部位发送至GPIO。

位0 将DBG\_POW\_STATE\_SWCORE的第0位发送至GPIO 34

位1 将DBG\_POW\_STATE\_SWCORE的第1位发送至GPIO 35

位2 将DBG\_POW\_STATE\_SWCORE的第2位发送至GPIO 36

1. +

2. + 位11将DBG\_POW\_STATE\_SWCORE的第11位信号输出至GPIO 45

表107。  
DBG\_POW\_OUTPUT\_TO\_GPIO寄存器

| 位     | 描述        | 类型 | 复位    |
|-------|-----------|----|-------|
| 31:12 | 保留。       | -  | -     |
| 11:0  | <b>启用</b> | 读写 | 0x000 |

## RP\_AP: IDR 寄存器

偏移量: 0xdfc

表 108. IDR  
寄存器

| 位    | 描述                | 类型 | 复位 |
|------|-------------------|----|----|
| 31:0 | 标准Coresight ID寄存器 | 只读 | -  |

## 3.6. Cortex-M33 协处理器

Cortex-M33 配备一个协处理器端口，能够在处理器与某些紧密耦合的硬件之间每个周期传输最多64位数据。Cortex-M33 内置的浮点单元即为此类协处理器的示例，RP2350 在该接口上新增了三个设备专用协处理器。以下章节对这些协处理器进行详尽说明。

在安全代码中访问协处理器之前，必须先通过设置 CPACR 中相应位启用该协处理器。非安全状态下访问时，须设置 NSA CR 和 CPACR\_NS 寄存器中的对应位。

RP2350 上的 RISC-V 处理器无法访问 Cortex-M33 协处理器。

### 3.6.1. GPIO 协处理器 (GPIOC)

协处理器端口 0 为 Cortex-M33 处理器对 SIO 中 GPIO 寄存器（第 3.1.3 节）提供低开销访问。该端口支持单条协处理器指令采样所有 48 个 GPIO，或设置、清除、写入任一单个 GPIO，及其他功能。

根据 ACCESSCTRL 中的 GPIO\_NSMASK0 和 GPIO\_NSMASK1 寄存器，非安全访问将被过滤。

未授权非安全使用的 GPIO 将忽略来自非安全状态的写入操作，且在非安全状态读取时返回零。

#### 3.6.1.1. OUT 掩码写入指令

这些指令写入 SIO GPIO\_OUT 和 GPIO\_HI\_OUT 寄存器中的多个位。

| 助记符                | Armv8-M 指令                           | 操作                                                                       |
|--------------------|--------------------------------------|--------------------------------------------------------------------------|
| gpioc_lo_out_put   | <code>mcr p0, #0, Rt, c0, c0</code>  | <code>sio_hw&gt;gpio_out = Rt;</code>                                    |
| gpioc_lo_out_xor   | <code>mcr p0, #1, Rt, c0, c0</code>  | <code>sio_hw&gt;gpio_togl = Rt;</code>                                   |
| gpioc_lo_out_set   | <code>mcr p0, #2, Rt, c0, c0</code>  | <code>sio_hw&gt;gpio_set = Rt;</code>                                    |
| gpioc_lo_out_clr   | <code>mcr p0, #3, Rt, c0, c0</code>  | <code>sio_hw&gt;gpio_clr = Rt;</code>                                    |
| gpioc_hi_out_put   | <code>mcr p0, #0, Rt, c0, c1</code>  | <code>sio_hw&gt;gpio_hi_out = Rt;</code>                                 |
| gpioc_hi_out_xor   | <code>mcr p0, #1, Rt, c0, c1</code>  | <code>sio_hw&gt;gpio_hi_togl = Rt;</code>                                |
| gpioc_hi_out_set   | <code>mcr p0, #2, Rt, c0, c1</code>  | <code>sio_hw&gt;gpio_hi_set = Rt;</code>                                 |
| gpioc_hi_out_clr   | <code>mcr p0, #3, Rt, c0, c1</code>  | <code>sio_hw&gt;gpio_hi_clr = Rt;</code>                                 |
| gpioc_hilo_out_put | <code>mcr p0, #0, Rt, Rt2, c0</code> | 同时： <code>sio_hw&gt;gpio_out = Rt; sio_hw&gt;gpio_hi_out = Rt2;</code>   |
| gpioc_hilo_out_xor | <code>mcr p0, #1, Rt, Rt2, c0</code> | 同时： <code>sio_hw&gt;gpio_togl = Rt; sio_hw&gt;gpio_hi_togl = Rt2;</code> |
| gpioc_hilo_out_set | <code>mcr p0, #2, Rt, Rt2, c0</code> | 同时： <code>sio_hw&gt;gpio_set = Rt; sio_hw&gt;gpio_hi_set = Rt2;</code>   |
| gpioc_hilo_out_clr | <code>mcr p0, #3, Rt, Rt2, c0</code> | 同时执行： <code>sio_hw&gt;gpio_clr = Rt; sio_hw&gt;gpio_hi_clr = Rt2;</code> |

#### 3.6.1.2. OE 掩码写入指令

这些指令用于写入 SIO GPIO\_OE 及 GPIO\_HI\_OE 寄存器的多个位。

| 助记符             | Armv8-M 指令                          | 操作                                           |
|-----------------|-------------------------------------|----------------------------------------------|
| gpioc_lo_oe_put | <code>mcr p0, #0, Rt, c0, c4</code> | <code>sio_hw&gt;gpio_oe = Rt;</code>         |
| gpioc_lo_oe_xor | <code>mcr p0, #1, Rt, c0, c4</code> | <code>sio_hw&gt;gpio_oe_togl = Rt;</code>    |
| gpioc_lo_oe_set | <code>mcr p0, #2, Rt, c0, c4</code> | <code>sio_hw&gt;gpio_oe_set = Rt;</code>     |
| gpioc_lo_oe_clr | <code>mcr p0, #3, Rt, c0, c4</code> | <code>sio_hw&gt;gpio_oe_clr = Rt;</code>     |
| gpioc_hi_oe_put | <code>mcr p0, #0, Rt, c0, c5</code> | <code>sio_hw&gt;gpio_hi_oe = Rt;</code>      |
| gpioc_hi_oe_xor | <code>mcr p0, #1, Rt, c0, c5</code> | <code>sio_hw&gt;gpio_hi_oe_togl = Rt;</code> |

| 助记符               | Armv8-M 指令                            | 操作                                                                                 |
|-------------------|---------------------------------------|------------------------------------------------------------------------------------|
| gpioc_hi_oe_set   | <code>mcr p0, #2, Rt, c0, c5</code>   | <code>sio_hw-&gt;gpio_hi_oe_set = Rt;</code>                                       |
| gpioc_hi_oe_clr   | <code>mcr p0, #3, Rt, c0, c5</code>   | <code>sio_hw-&gt;gpio_hi_oe_clr = Rt;</code>                                       |
| gpioc_hilo_oe_put | <code>mcrr p0, #0, Rt, Rt2, c4</code> | 同时执行: <code>sio_hw-&gt;gpio_oe = Rt; sio_hw-&gt;gpio_hi_oe = Rt2;</code>           |
| gpioc_hilo_oe_xor | <code>mcrr p0, #1, Rt, Rt2, c4</code> | 同时执行: <code>sio_hw-&gt;gpio_oe_togl = Rt; sio_hw-&gt;gpio_hi_oe_togl = Rt2;</code> |
| gpioc_hilo_oe_set | <code>mcrr p0, #2, Rt, Rt2, c4</code> | 同时执行: <code>sio_hw-&gt;gpio_oe_set = Rt; sio_hw-&gt;gpio_hi_oe_set = Rt2;</code>   |
| gpioc_hilo_oe_clr | <code>mcrr p0, #3, Rt, Rt2, c4</code> | 同时执行: <code>sio_hw-&gt;gpio_oe_clr = Rt; sio_hw-&gt;gpio_hi_oe_clr = Rt2;</code>   |

### 3.6.1.3 单位比特写入指令

这些指令写入GPIO\_OUT和GPIO\_HI\_OUT寄存器，或GPIO\_OE和GPIO\_HI\_OE寄存器中单一索引位。

| 助记符                | Armv8-M 指令                            | 操作                                                                                                                              |
|--------------------|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| gpioc_bit_out_put  | <code>mcrr p0, #4, Rt, Rt2, c0</code> | 向任意输出写入单比特值。等同于: <code>if (Rt2 &amp; 1) gpioc_hilo_out_set(1ull &lt;&lt; Rt); else gpioc_hilo_out_clr(1ull &lt;&lt; Rt);</code> |
| gpioc_bit_out_xor  | <code>mcr p0, #5, Rt, c0, c0</code>   | 无条件切换任意单个输出。等同于:<br><code>gpioc_hilo_out_xor(1ull &lt;&lt; Rt);</code>                                                          |
| gpioc_bit_out_set  | <code>mcr p0, #6, Rt, c0, c0</code>   | 无条件设置任意单个输出。等同于:<br><code>gpioc_hilo_out_set(1ull &lt;&lt; Rt);</code>                                                          |
| gpioc_bit_out_clr  | <code>mcr p0, #7, Rt, c0, c0</code>   | 无条件清除任意单个输出。等同于:<br><code>gpioc_hilo_out_clr(1ull &lt;&lt; Rt);</code>                                                          |
| gpioc_bit_out_xor2 | <code>mcrr p0, #5, Rt, Rt2, c0</code> | 有条件切换任意单个输出。等同于:<br><code>gpioc_hilo_out_xor((uint64_t)(Rt2 &amp; 1) &lt;&lt; Rt);</code>                                       |
| gpioc_bit_out_set2 | <code>mcrr p0, #6, Rt, Rt2, c0</code> | 有条件设置任意单个输出。等同于:<br><code>gpioc_hilo_out_set((uint64_t)(Rt2 &amp; 1) &lt;&lt; Rt);</code>                                       |
| gpioc_bit_out_clr2 | <code>mcrr p0, #7, Rt, Rt2, c0</code> | 有条件清除任意单个输出。等同于:<br><code>gpioc_hilo_out_clr((uint64_t)(Rt2 &amp; 1) &lt;&lt; Rt);</code>                                       |
| gpioc_bit_oe_put   | <code>mcrr p0, #4, Rt, Rt2, c4</code> | 向任意输出使能写入1位值。等同于: <code>if (Rt2 &amp; 1) gpioc_hilo_oe_set(1ull &lt;&lt; Rt); else gpioc_hilo_oe_clr(1ull &lt;&lt; Rt);</code>  |
| gpioc_bit_oe_xor   | <code>mcr p0, #5, Rt, c0, c4</code>   | 无条件切换任意输出使能。等同于:<br><code>gpioc_hilo_oe_xor(1ull &lt;&lt; Rt);</code>                                                           |
| gpioc_bit_oe_set   | <code>mcr p0, #6, Rt, c0, c4</code>   | 无条件设置任意输出使能（设为输出）。等同于:<br><code>gpioc_hilo_oe_set(1ull &lt;&lt; Rt);</code>                                                     |
| gpioc_bit_oe_clr   | <code>mcr p0, #7, Rt, c0, c4</code>   | 无条件清除任意输出使能（设为输入）。等同于:<br><code>gpioc_hilo_oe_clr(1ull &lt;&lt; Rt);</code>                                                     |
| gpioc_bit_oe_xor2  | <code>mcrr p0, #5, Rt, Rt2, c4</code> | 有条件地切换任意输出使能。等同于:<br><code>gpioc_hilo_oe_xor((uint64_t)(Rt2 &amp; 1) &lt;&lt; Rt);</code>                                       |
| gpioc_bit_oe_set2  | <code>mcrr p0, #6, Rt, Rt2, c4</code> | 有条件地设置任意输出使能（设为输出）。等同于:<br><code>gpioc_hilo_oe_set((uint64_t)(Rt2 &amp; 1) &lt;&lt; Rt);</code>                                 |
| gpioc_bit_oe_clr2  | <code>mcrr p0, #7, Rt, Rt2, c4</code> | 有条件地清除任意输出使能（设为输入）。等同于:<br><code>gpioc_hilo_oe_clr((uint64_t)(Rt2 &amp; 1) &lt;&lt; Rt);</code>                                 |

### 3.6.1.4. 索引掩码写入指令

这些指令写入单个动态选择的32位GPIO寄存器。

| 助记符                 | Armv8-M 指令                              | 操作                                                                   |
|---------------------|-----------------------------------------|----------------------------------------------------------------------|
| gpioc_index_out_put | <code>mcr r p0, #8, Rt, Rt2, c0</code>  | 将 <code>Rt</code> 写入由 <code>Rt2</code> 选择的 GPIO 输出寄存器。               |
| gpioc_index_out_xor | <code>mcr r p0, #9, Rt, Rt2, c0</code>  | 切换由 <code>Rt2</code> 选择的 GPIO 输出寄存器中位于 <code>Rt</code> 的位置的位。        |
| gpioc_index_out_set | <code>mcr r p0, #10, Rt, Rt2, c0</code> | 设置由 <code>Rt2</code> 选择的 GPIO 输出寄存器中位于 <code>Rt</code> 的位置的位。        |
| gpioc_index_out_clr | <code>mcr r p0, #11, Rt, Rt2, c0</code> | 清除由 <code>Rt2</code> 选择的 GPIO 输出寄存器中位于 <code>Rt</code> 的位置的位。        |
| gpioc_index_oe_put  | <code>mcr r p0, #8, Rt, Rt2, c4</code>  | 将 <code>Rt</code> 写入由 <code>Rt2</code> 选择的 GPIO 输出使能寄存器。             |
| gpioc_index_oe_xor  | <code>mcr r p0, #9, Rt, Rt2, c4</code>  | 切换由 <code>Rt2</code> 选择的 GPIO 输出使能寄存器中位于 <code>Rt</code> 的位置的位。      |
| gpioc_index_oe_set  | <code>mcr r p0, #10, Rt, Rt2, c4</code> | 设置位 <code>Rt</code> 于由 <code>Rt2</code> 选择的 GPIO 输出使能寄存器中（即设置为输出状态）。 |
| gpioc_index_oe_clr  | <code>mcr r p0, #11, Rt, Rt2, c4</code> | 清除位 <code>Rt</code> 于由 <code>Rt2</code> 选择的 GPIO 输出使能寄存器中（即设置为输入状态）。 |

### 3.6.1.5. 读取指令

这些指令从GPIO\_OUT与GPIO\_HI\_OUT寄存器读取；GPIO\_OE与GPIO\_HI\_OE寄存器；或GPIO\_IN与GPIO\_HI\_IN寄存器。

| 助记符                | Armv8-M 指令                            | 操作                                                                                                                            |
|--------------------|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| gpioc_lo_out_get   | <code>mrc p0, #0, Rt, c0, c0</code>   | 读取低32位输出寄存器的值。等价于： <code>Rt = sio_hw-&gt;gpio_out;</code>                                                                     |
| gpioc_hi_out_get   | <code>mrc p0, #0, Rt, c0, c1</code>   | 读取高32位输出寄存器的值。等价于： <code>Rt = sio_hw-&gt;gpio_hi_out;</code>                                                                  |
| gpioc_hilo_out_get | <code>mrrc p0, #0, Rt, Rt2, c0</code> | 在一次操作中读回两个32位输出寄存器。<br>等同于： <code>Rt = sio_hw-&gt;gpio_out;</code> 并同时 <code>Rt2 = sio_hw-&gt;gpio_hi_out &lt;&lt; 32;</code> |
| gpioc_lo_oe_get    | <code>mrc p0, #0, Rt, c0, c4</code>   | 读回低32位输出使能寄存器。等同于： <code>Rt = sio_hw-&gt;gpio_oe;</code>                                                                      |
| gpioc_hi_oe_get    | <code>mrc p0, #0, Rt, c0, c5</code>   | 读回高32位输出使能寄存器。等同于： <code>Rt = sio_hw-&gt;gpio_hi_oe;</code>                                                                   |
| gpioc_hilo_oe_get  | <code>mrrc p0, #0, Rt, Rt2, c4</code> | 在一次操作中读回两个32位输出使能寄存器。<br>等同于： <code>Rt = sio_hw-&gt;gpio_oe;</code> 并同时 <code>Rt2 = sio_hw-&gt;gpio_hi_oe &lt;&lt; 32;</code> |
| gpioc_lo_in_get    | <code>mrc p0, #0, Rt, c0, c8</code>   | 采样低 32 个 GPIO。等同于： <code>Rt = sio_hw-&gt;gpio_in;</code>                                                                      |
| gpioc_hi_in_get    | <code>mrc p0, #0, Rt, c0, c9</code>   | 采样高 32 个 GPIO。等同于： <code>Rt = sio_hw-&gt;gpio_hi_in;</code>                                                                   |
| gpioc_hilo_in_get  | <code>mrrc p0, #0, Rt, Rt2, c8</code> | 在同一周期采样 64 个 GPIO。等同于： <code>Rt = sio_hw-&gt;gpio_in;</code> 且同时 <code>Rt2 = sio_hw-&gt;gpio_hi_in &lt;&lt; 32;</code>        |

### 3.6.1.6. 指令字段的解析

协处理器指令类型 – `mrc`、`mrrc`、`mcr` 和 `mrr` – 指明传输方向（读/写）及所传输的 Arm 寄存器数量（一或两个）。

第一个协处理器寄存器号字段CRm的第 3 至 2 位指定所访问的寄存器组。数值 0、1 及

2 分别指向输出寄存器、输出使能寄存器和输入寄存器。

第一个协处理器寄存器编号字段CRm的第0位可用于区分正在访问的寄存器组中的具体寄存器。第1位予以保留，以便未来拥有更多GPIO的芯片能够索引更多寄存器。

对于写操作，指令的opc1字段的第1:0位指定写入操作的类型：数值0、1、2、3分别对应普通写、异或（XOR）、置位和清除操作。opc1字段的第3:2位用于指示所访问寄存器或单个位的寻址模式。该字段的具体含义依赖于指令本身的定义。

前述表格未列举的所有组合均保留供将来使用。

### 3.6.2. 双精度协处理器（DCP）

每个Cortex-M33 CPU内核均配备两个双精度协处理器实例，用于加速包括加法、减法、乘法、除法和平方根在内的双精度浮点运算。该设计仅由几千个逻辑门实现，因此占用的硅晶片面积远小于完整的双精度浮点单元。

尽管如此，这些协处理器相比纯软件实现，大幅加速了基本双精度运算。协处理器还支持部分单精度运算及相应的转换。

这两个协处理器实例分别分配至安全域与非安全域。协处理器编号 4始终映射至当前处理器安全状态所对应的协处理器。协处理器编号 5始终映射至非安全协处理器实例，但仅限安全代码访问。此重复配置避免了安全/非安全状态切换时协处理器上下文的保存与恢复。

#### 3.6.2.1. CPU接口

与其他协处理器一样，加速器通过64位总线连接CPU。可通过以下指令在该总线上每周期传输两个字的数据：

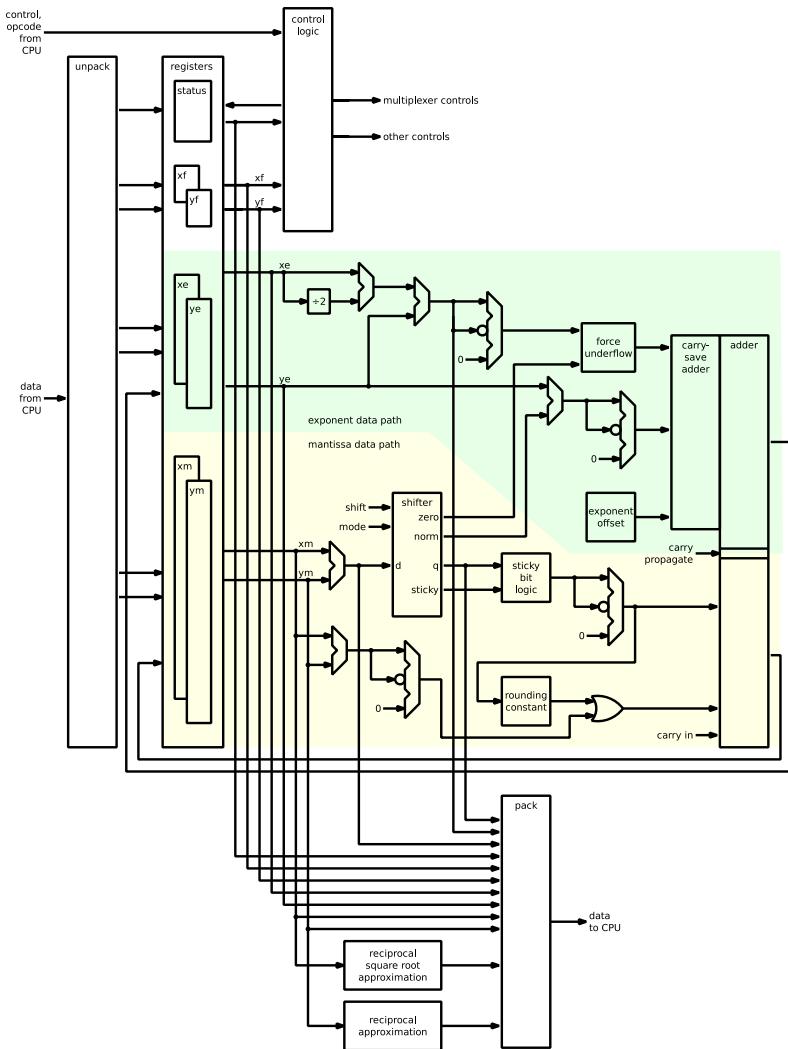
- **MCRR**：将两个整数寄存器传送到协处理器
- **MRRC**：从协处理器传送两个整数寄存器

这些指令还有单寄存器版本，包括允许从协处理器加载CPU标志的指令。CPU发出 **COP** 指令以触发协处理器内的操作，且不传输任何数据。

#### 3.6.2.2 内部架构

加速器的框图如图12所示。

图12。双精度  
加速器框图



设计核心包括：

- 两组寄存器，每组设计用于存放一个拆包的双精度数值
- 一个9位状态寄存器

与传统的FPU不同，加速器不包含完整的寄存器组。这不仅节省了芯片面积，而且意味着保存和恢复协处理器状态非常快速：实际上，整个状态仅占六个32位字，因而可通过三条指令保存到CPU或从CPU恢复。

该加速器包含一台宽加法器，能够同时对两个尾数值和三个指数值进行加法运算。此外，还配备了一台移位器，可执行逻辑右移指定位数，或对非规格化尾数进行规格化并报告所需的移位量。移位器的相当部分硬件在这两种操作模式之间共享。

图示顶部的控制逻辑解码协处理器指令，并配置加速器的功能单元和数据路径多路复用器，以执行指定操作。每条协处理器指令占用单个周期，因此协处理器操作不会导致CPU停顿。

执行浮点运算（例如加法或减法）时，采用以下固定（或称“预设”）指令序列：

1. 一条或两条 **MCRR** 指令，用于向协处理器写入操作数。
2. 若干条 **CDP** 指令（以及可能的其他指令）共同完成运算操作。
3. 一个 **MRRC** 或 **MRC** 指令，用于读取返回结果。

硬件处理涉及零值、NaN（非数字）和无穷大等特殊情况，以及舍入、下溢和上溢情况。

该加速器不包含乘法器阵列，因为那将占用相当大的一块芯片面积。相反，乘法操作数的尾数被送回CPU，以利用CPU中可用的高速长乘指令。协处理器负责指数部分的处理。

除法和平方根操作同样涉及协处理器与CPU之间的数据来回传输。为辅助这些操作，协处理器包含两个小型查找表（以随机逻辑实现），用于提供除法和平方根算法中的初始近似值。协处理器负责指数部分的处理。

该加速器仅用于执行实现基本浮点运算的预设指令序列。加速器状态不保证从一个预设序列结束时至下一个序列开始时保持不变：详见下文状态寄存器中“engaged”标志的说明。

### 3.6.2.3. 寄存器

#### X 和 Y 尾数寄存器

X 和 Y 尾数寄存器 ( $xm$  和  $ym$ ) 各为 64 位宽。它们可以被 CPU 直接读取和写入；  
 $xm$  寄存器还可存储加法器结果的低位部分。当使用“写入非打包” MCRR 指令向协处理器写入值时，尾数寄存器的最高两位被设为 01，次高位从浮点操作数的尾数字段填充。尾数寄存器的低位被清零。

#### X 和 Y 指数寄存器

X 和 Y 指数寄存器 ( $xe$  和  $ye$ ) 各为 14 位宽。它们可以被 CPU 直接读取和写入；  
 $xe$  寄存器还可存储加法器结果的高位部分。当使用“写入非打包” MCRR 指令向协处理器写入值时，指数寄存器从浮点操作数的指数组字段设定。

#### X 和 Y 标志寄存器

X 和 Y 标志寄存器 ( $xf$  和  $yf$ ) 各为四位宽。它们可被 CPU 直接读取和写入。标志寄存器存储有关对应尾数寄存器和指数寄存器中所表示浮点数类型的信息：其符号、是否为零、是否为无穷大以及是否为 NaN。当使用“写入未封装” MCRR 指令向协处理器写入值时，标志寄存器的各位将根据浮点操作数类型进行更新。

#### 状态寄存器

状态寄存器包含九位。它可被 CPU 直接读取和写入。寄存器的最低六位存储对两个加法或减法操作数进行对齐所需的移位数；接下来的两位指示由 ( $xe$ ,  $xm$ ) 表示的值是否大于、等于或小于由 ( $ye$ ,  $ym$ ) 表示的值——即指示存储于 X 寄存器中的值的幅度是否大于、等于或小于存储于 Y 寄存器中的值的幅度。这些状态位在加法、减法或比较操作的第一步中，于操作数加载后被设置。

状态寄存器的最后一位指示协处理器是否处于“启用”状态。所有位于预设指令序列开头或中间的协处理器指令均会设置启用标志。该标志会被用于读取最终结果、处于预设序列末尾的指令清除。

### 3.6.2.4. 状态保存与恢复

中断处理程序可以检测启用标志，以确认是否抢占了同一协处理器上正在进行的操作。若启用标志被设置，处理程序可在使用协处理器前保存（及恢复）协处理器状态。若启用标志被清除，则可跳过保存（及恢复）步骤。若采用此方法，则在不处于任何预设指令序列内时，必须将加速器状态视为未定义。

提供三条 MRRC 指令，用于将协处理器中的六个状态字复制到 CPU 的整数寄存器中，  
 例如，可以将其推入堆栈。其中最后一条指令用于清除 engaged 标志。

类似地，提供三条 MCRR 指令，用于从整数寄存器恢复协处理器状态，包括 engaged 标志的状态。

### 3.6.2.5. 指令概要

如前所述，协处理器指令仅应作为预定义序列的一部分使用。

然而，为完整起见，现列出可用指令及其功能概要。

MCRR 指令详见表109。

表109. MCRR  
指令

| 助记符  | 效果                                           | 使用者      |
|------|----------------------------------------------|----------|
| WXMD | 写入 $xm$ 直接                                   | 恢复状态     |
| WYMD | 写入 $ym$ 直接                                   | 恢复状态     |
| WEFD | 写入 $xe$ 、 $xf$ 、 $ye$ 、 $yf$ 及其他状态（直接）       | 恢复状态     |
| WXUP | write $xm, xe, xf$ 拆包双精度                     | 双精度二进制运算 |
| WYUP | write $ym, ye, yf$ 拆包双精度                     | 双精度二进制运算 |
| WXYU | write $xm, xe, xf, ym, ye, yf$ 两个拆包单精度       | 单精度二进制运算 |
| WXMS | write $xm$ 第0位=0/1，表示数据为零/非零                 | dmul     |
| WXMO | write $xm$ 直接对b0进行 OR 运算，指数相加，符号执行 XOR       | dmul     |
| WXDD | write $xm$ 直接写入；指数相减，符号执行 XOR                | ddiv     |
| WDXQ | 写入 $xm$ 直接，偏移指数                              | dsqrt    |
| WXUC | 写入 X 无符号整数 $+2^{52}+2^{32}$ , $Y=252+2^{32}$ | 从无符号整数转换 |
| WXIC | 写入 X 有符号整数 $+2^{52}+2^{32}$ , $Y=252+2^{32}$ | 从有符号整数转换 |
| WXOC | 写入 X 未打包双精度， $Y=2^{52}+2^{32}$               | 从双精度转换   |
| WXFC | 写入 X 未打包单精度， $Y=2^{52}+2^{32}$               | 从单精度转换   |
| WXML | 写入 $xm$ 直接，指数相加，符号异或                         | fmul     |
| WXFD | 写入 $xm$ 直接，指数相减，异或符号                         | fdiv     |
| WXFQ | 写入 $xm$ 直接，偏移指数                              | fsqrt    |

CDP 指令见表110。

表110. CDP  
指令

| 助记符  | 效果                                          | 使用者    |
|------|---------------------------------------------|--------|
| 初始化  | 清零所有寄存器                                     |        |
| ADD0 | 比较 $X-Y$ ，设置状态                              | 加、减、比较 |
| ADD1 | $xm:=\pm xm+\pm ym>>s$ 或 $\pm ym+\pm xm>>s$ | 加      |
| SUB1 | $xm:=\pm xm-\pm ym>>s$ 或 $-\pm ym\pm xm>>s$ | 减      |
| SQR0 | $xe=xe/2$ , $xm=xm<<0:1$                    | sqrt   |

| 助记符  | 效果                  | 使用者                   |
|------|---------------------|-----------------------|
| NORM | 归一化                 |                       |
| NRDF | 归一化并对单精度值进行四舍五入     | 单精度<br>运算, 转换<br>为单精度 |
| NRDD | 归一化并对双精度值进行四舍五入     | 双精度<br>运算, 转换<br>为双精度 |
| NTDC | 归一化并截断双精度整数前转换      | 截断转换为<br>int          |
| NRDC | 归一化并对双精度整数前转换进行四舍五入 | 四舍五入转换为<br>int        |

**MRRC** 和 **MRC** 指令详见表111。

表111。MRRC和  
MRC指令

| 助记符  | 效果                                                        | 使用者               |
|------|-----------------------------------------------------------|-------------------|
| RXVD | 读取 <b>xf</b> , VERSION 直接                                 | dclassify, 检查版本   |
| RCMP | 读取处理状态                                                    | dcmp              |
| RDFA | 从 X 中读取打包的 FADD 结果                                        | fadd              |
| RDFS | 从 X 中读取打包的 FSUB 结果                                        | fsub              |
| RDFM | 从 X 中读取打包的 FMUL 结果                                        | fmul              |
| RDFD | 从 X 中读取打包的 FDIV 结果                                        | fdiv              |
| RDFQ | 从 X 中读取打包的 FSQRT 结果                                       | fsqrt             |
| RDFG | 从 X 读取通用浮点数结果 (打包)                                        | 双精度转换为单精度         |
| RDUC | 从 X 读取无符号整数转换结果                                           | 转换为无符号整数          |
| RDIC | 从 X 读取有符号整数转换结果                                           | 转换为有符号整数          |
| RXMD | 读取 <b>xm</b> 直接                                           | 保存状态              |
| RYMD | 读取 <b>ym</b> 直接, engaged=0                                | 保存状态              |
| REFD | 读取 <b>xe</b> , <b>xf</b> , <b>ye</b> , <b>yf</b> , 其他状态直接 | 保存状态              |
| RXMS | 读取 <b>xm</b> Q62-s                                        | dmul, ddiv, dsqrt |
| RYMS | 读 <b>ym</b> Q62-s                                         | dmul, ddiv        |
| RXYH | 读 <b>ym</b> hi, <b>xm</b> hi                              | fmul, fdiv        |
| RYMR | 读 <b>ym</b> hi, 倒数近似低位                                    | fdiv, ddiv        |
| RXMQ | 读 <b>xm</b> hi, 平方根倒数近似低位                                 | fsqrt, dsqrt      |
| RDDA | 读取从 X 打包的 DADD 结果                                         | dadd              |
| RDDS | 读取从 X 打包的 DSUB 结果                                         | dsub              |
| RDDM | 从 X 读取 DMUL 结果的打包数据                                       | dmul              |
| RDDD | 从 X 读取 DDIV 结果的打包数据                                       | ddiv              |

| 助记符  | 效果               | 使用者       |
|------|------------------|-----------|
| RDDQ | 从X读取DSQRT结果的打包数据 | dsqrt     |
| RDDG | 从X读取通用双精度结果的打包数据 | 单精度到双精度转换 |

每个 **MRRC** 和 **MRC** 指令旁均有一个变体，使用 **P**（代表“peek”）替代 **R**，功能相同但保留已置位的标志。**RXMD** 与 **PXMD** 完全相同；**REFD** 与 **PEFD** 完全相同。

SDK 包含宏，可根据上述助记符生成 Arm 汇编，宏定义位于 SDK 中的文件 **dcp\_instr.inc.S** 中，例如，将 **WXUP r0,r1** 转换为 **mcrr p4,#1,r0,r1,c0**。

### 3.6.2.6. 示例预设序列

表 112 展示了实现可调用双精度加法操作的汇编代码序列。

表 112。实现可调用双精度加法操作的汇编代码序列

| Arm 汇编                       | 协处理器助记符           | 操作                                  |
|------------------------------|-------------------|-------------------------------------|
| <b>mcrr p4,#1,r0,r1,c0</b>   | <b>WXUP r0,r1</b> | 将拆包的双精度数 R0 和 R1 写入 X               |
| <b>mcrr p4,#1,r2,r3,c1</b>   | <b>WYUP r2,r3</b> | 将拆包的双精度数 R2 和 R3 写入 Y               |
| <b>cdp p4,#0,c0,c0,c1,#0</b> | <b>ADD0</b>       | 比较 X 和 Y；设置状态和对齐移位                  |
| <b>cdp p4,#1,c0,c0,c1,#0</b> | <b>ADD1</b>       | 根据状态及符号执行加/减操作，xm 和 ym 对齐，结果写入 xm   |
| <b>cdp p4,#8,c0,c0,c0,#1</b> | <b>NRDD</b>       | 归一化并四舍五入双精度结果                       |
| <b>mrrc p4,#1,r0,r1,c0</b>   | <b>RDDA r0,r1</b> | 从 X 读取 R0 与 R1 打包的双精度值，包括针对加法的特殊值处理 |
| <b>bx r14</b>                |                   | 函数返回                                |

协处理器中的逻辑确保，例如，**ADD1** 指令会对较小的参数进行移位，且在送入加法器之前，**xm** 和 **ym** 会根据要求被取反，同时使用较大的指数作为后续规格化的基础。

### 3.6.2.7. 通过 SDK 库使用协处理器

SDK **pico\_double** 库会自动调用协处理器以执行双精度浮点运算。这是利用协处理器的最简便方式，但每个操作都会带来数个周期的开销。不仅存在函数调用和返回的开销，且为保证安全，SDK 中的通用实现总是检测使能标志，并根据需要将协处理器状态保存至堆栈或从堆栈恢复。此措施确保若在中断处理程序中使用函数，其功能能正确执行，无需额外介入。

### 3.6.2.8. 直接使用协处理器

SDK 包含用于在文件 **dcp\_canned.inc.S**（位于 SDK 中）生成标准操作预定义序列的宏。

例如，上述可调用双精度加法操作可写为：

```
dcp_dadd_m r0, r1, r0, r1, r2, r3 @ 结果存放于 r0, r1；操作数位于 r0, r1 和 r2, r3
bx r14
```

`dcp_dadd_m`为宏，展开后形成上述协处理器指令序列。该宏允许指定用于操作数和结果的整型寄存器，既避免函数调用与返回开销，又消除参数封送处理的额外负担。

较复杂的宏还要求指定‘临时’寄存器，用于保存中间结果。以下函数通过计算由 `R0` 和 `R1` 指向的两个三元素双精度向量的点积予以说明：

```

push {r4-r9, r14}
lrd r3, r4, [r0], #8 @ 加载 x0
lrd r5, r6, [r1], #8 @ 加载 y0
dcp_dmull_m r7, r8, r3, r4, r5, r6, r3, r4, r5, r6, r12, r14, r9 @ 计算 x0y0 ①
lrd r3, r4, [r0], #8 @ 加载 x1
lrd r5, r6, [r1], #8 @ 加载 y1
dcp_dmull_m r3, r4, r3, r4, r5, r6, r3, r4, r5, r6, r12, r14, r9 @ 计算 x1y1 ①
dcp_dadd_m r7, r8, r3, r4, r7, r8 @ 计算 x0y0+x1y1
lrd r3, r4, [r0], #8 @ 加载 x2
lrd r5, r6, [r1], #8 @ 加载 y2
dcp_dmull_m r3, r4, r3, r4, r5, r6, r3, r4, r5, r6, r12, r14, r9 @ 计算 x2y2 ①
dcp_dadd_m r0, r1, r3, r4, r7, r8 @ 计算 x0y0+x1y1+x2y2②
出栈 {r4-r9, r15}

```

1. `r3`、`r4`、`r5`、`r6`、`r12`、`r14` 及 `r9` 为临时寄存器。

2. 结果存储于 `r0`、`r1`。

### 注意

该示例未检查激活标志。若在中断处理程序或多线程应用中使用，须添加适当检测。例如，参阅 SDK 中 `_aeabi_dadd` 的实现，以获得高效的方法。该测试仅需在函数开始时执行一次，因此在此情况下的开销相对较小。

以下示例演示如何使用协处理器：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/dcp/hello\\_dcp/hello\\_dcp.c](https://github.com/raspberrypi/pico-examples/blob/master/dcp/hello_dcp/hello_dcp.c) 第 18 行至第 109 行

```

18 extern double dcp_dot (double*p, double*q, int n);
19 extern double dcp_dotx (float*p, float*q, int n);
20 extern float dcp_iirx (float x, float*temp, float*coeff, int order);
21 extern void dcp_butterfly_radix2 (double*x, double*y);
22 extern void dcp_butterfly_radix2_twiddle_dif (double*x, double*y, double*tf);
23 extern void dcp_butterfly_radix2_twiddle_dit (double*x, double*y, double*tf);
24 extern void dcp_butterfly_radix4 (double*w, double*x, double*y, double*z);
25
26 static void dcp_test0() {
27 double u[3]={1,2,3};
28 double v[3]={4,5,6};
29 double w;
30 w=dcp_dot(u,v,3);
31 printf("(1,2,3).(4,5,6)=%g\n",w);
32 }
33
34 static void dcp_test1() {
35 float u[3]={1+pow(2,-20),2,3};
36 float v[3]={1-pow(2,-20),5,6};
37 double w;
38 w=dcp_dotx(u,v,3);

```

```

39 printf("(1+pow(2,-20),2,3).(1-pow(2,-20),5,6)=%.17g\n",w);
40 }
41
42 static void dcp_test2() {
43 int t;
44 float w;
45 // filter coefficients calculated using Octave as follows:
46 // octave> pkg load signal
47 // octave> format long
48 // octave> [b,a]=cheby1(2,1,.5)
49 // b = 0.307043201259064 0.614086402518128 0.307043201259064
50 // a = 1.00000000000000e+00 6.406405700380895e-02 3.139684953186774e-01
51 // 并按如下方式进行测试:
52 // octave> filter(b,a,[1 zeros(1,19)])
53 float coeff[5]={0.3070432,0.3139685,0.6140864,0.06406406,0.3070432};
54 float temp[4]={0};
55 printf("IIR滤波器脉冲响应:\n");
56 for(t=0;t<20;t++) {
57 w=dcp_iirx(t?0:1,temp,coeff,2);
58 printf("y[%d]=%g\n",t,w);
59 }
60 }
61
62 static void dcp_test3() {
63 double x[2]={2,3};
64 double y[2]={5,7};
65 dcp_butterfly_radix2(x,y);
66 printf("(2+3j,5+7j)的基数2蝶形变换=(%g%+gj,%g%+gj)\n",x[0],x[1],y[0],y[1]);
67 }
68
69 static void dcp_test4() {
70 double x[2]={2,3};
71 double y[2]={5,7};
72 double t[2]={1.5,2.5};
73 dcp_butterfly_radix2_twiddle_dif(x,y,t);
74 printf("带旋转因子(1.5+2.5j)的(2+3j,5+7j)基数2DIF蝶形变换=(%g%+gj,%g
75 %+gj)\n",x[0],x[1],y[0],y[1]); 75 }

76
77 static void dcp_test5() {
78 double x[2]={2,3};
79 double y[2]={5,7};
80 double t[2]={1.5,2.5};
81 dcp_butterfly_radix2_twiddle_dit(x,y,t);
82 printf("基数-2 DIT蝶形变换 (2+3j,5+7j) 乘以旋转因子 (1.5+2.5j)=(%g%+gj
83 ,%g%+gj)\n",x[0],x[1],y[0],y[1]); 83 }

84
85 static void dcp_test6() {
86 double w[2]={2,3};
87 double x[2]={5,7};
88 double y[2]={11,17};
89 double z[2]={41,43};
90 dcp_butterfly_radix4(w,x,y,z);
91 printf("基数-4 蝶形变换 (2+3j,5+7j,11+17j,41+43j) = (%g%+gj,%g%+gj,%g%+gj,%g%+gj)\n",w[0],w
92 [1],x[0],x[1],y[0],y[1],z[0],z[1]); 92 }

93
94 int main() {
95 stdio_init_all();
96
97 printf("Hello, DCP!\n");
98
99 dcp_test0();

```

```

100 dcp_test1();
101 dcp_test2();
102 dcp_test3();
103 dcp_test4();
104 dcp_test5();
105 dcp_test6();
106
107 return 0;
108 }
```

在 Pico Examples 仓库的 `dcp/` 目录中还有更多示例。

### 3.6.2.9. IEEE 754 合规性

预置指令序列提供符合 IEEE 标准的操作，但输入和输出时非规格化数会被冲零处理。零值、NaN 和无穷大均得到正确处理。舍入到最接近值，遇到平分则舍入至偶数。

提供名为 `ddiv_fast` 和 `dsqrt_fast` 的加速版除法及平方根操作。该版本的结果不总是精确舍入，但保证舍入前误差小于  $0.5u$  Ip（单位最后一位），特别是若结果存在精确表示，则返回该精确值。

### 3.6.2.10. 基准测试

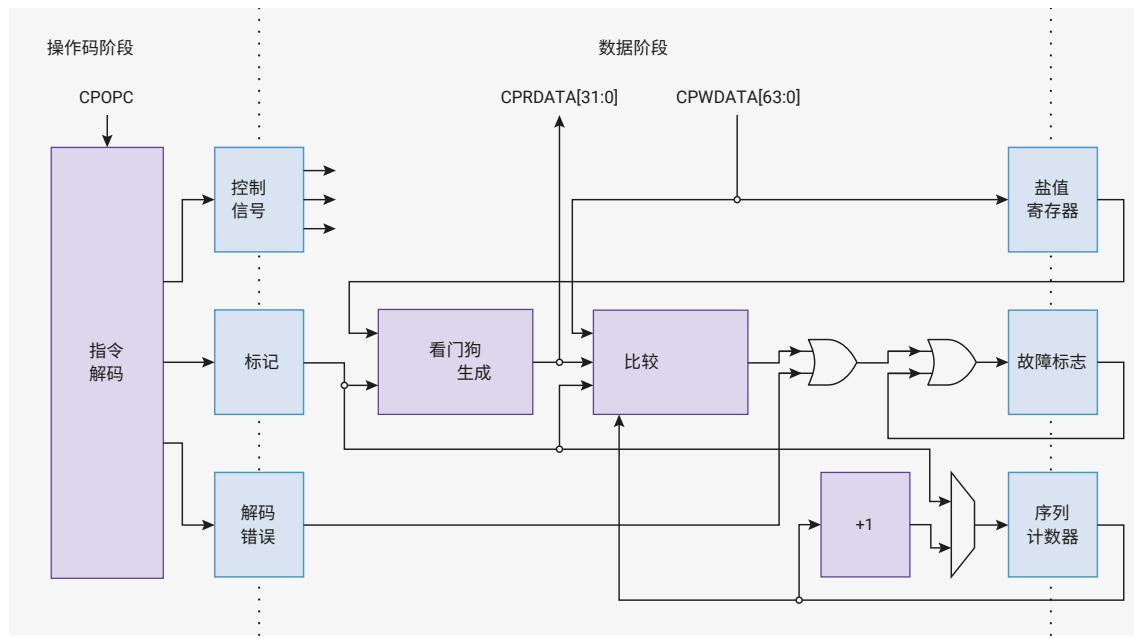
表113列出了使用加速器内联代码执行各种浮点运算的周期数，且将其与（a）完整硬件双精度FPU的典型基准范围进行了比较；以及（b）纯软件实现。

表113。使用  
加速器执行浮点  
运算的周期数

| 操作                          | 使用<br>协处理器 | 完整硬件（延迟） | 仅软件     |
|-----------------------------|------------|----------|---------|
| <code>dadd</code>           | 6          | 2-6      | 70-90   |
| <code>dsub</code>           | 6          | 2-6      | 70-90   |
| <code>dmul</code>           | 17         | 3-7      | 75-90   |
| <code>ddiv</code>           | 51         | 13-60    | 135-600 |
| <code>ddiv_fast</code>      | 32         |          |         |
| <code>dsqrt</code>          | 49         | 15-62    | 130-650 |
| <code>dsqrt_fast</code>     | 38         |          |         |
| <code>dcmp</code>           | 4          |          |         |
| <code>declassify</code>     | 2          |          |         |
| 整数与 <code>double</code> 的转换 | 5          |          |         |

### 3.6.3. 冗余协处理器（RCP）

图13。冗余  
协处理器  
实现硬件  
校验断言，以辅助  
控制流程和数据流  
程的完整性检查  
。其两阶段流  
水线与Cortex-M  
33流水线紧密  
耦合。一个64位盐  
寄存器保存每次启  
动的随机数，该  
随机数用于生成和  
校验堆栈金丝雀值  
，并在RCP指令  
中生成伪随机延  
迟序列。其他  
比较功  
能提供更通用的硬  
件校验断言支持  
。



冗余协处理器（RCP）用于RP2350只读存储器启动程序中，提供针对故障注入及面向返回编程攻击的硬件辅助缓解措施。包括以下指令：

- 基于每次启动的随机种子生成和校验堆栈金丝雀值
- 断言程序中某些点按正确顺序执行且无遗漏步骤
- 校验存储于32位字中、以两种有效位模式之一表示的布尔值
- 验证以异或奇偶校验掩码冗余存储于两个字中的32位整数
- 在检测到软件触发的紧急状况时，停止处理器运行

第3.6.3.7节完整列出了RCP指令集。RCP指令编码包含奇偶校验位；执行无效指令或奇偶校验错误的指令将触发RCP故障。

每个Cortex-M33处理器配备一个RCP实例，映射为协处理器操作码空间中的协处理器编号 7。两个RCP实例相互连接：一个内核发生RCP故障时，立即触发另一内核故障。RCP故障包含两个步骤：

1. 不可屏蔽中断（NMI）被激活。该中断保持激活状态，直至处理器完成热复位。
2. 所有后续RCP指令将使协处理器端口阻塞，直到处理器完成热复位。该阻塞状态不可中断，因处理器已处于NMI状态。

RP2350 只读存储器实现了带有 `rcp_panic` 指令的 NMI 和 HardFault 向量。该指令无条件阻塞协处理器端口。这阻止处理器完成更多指令，直到调试器连接以重置处理器，或处理器通过其他机制（例如系统看门狗定时器）重置。处理器迅速进入静止状态，从而降低了对进一步故障注入（无论故意与否）的脆弱性。

每个核心的RCP均有一个64位种子值（第3.6.3.1节）。RCP使用该值生成栈金丝雀值，并向RCP指令添加短暂的伪随机延迟。两个RCP实例均由核心0在启动只读存储器早期引导路径中，通过系统真随机数生成器（第12.12节）初始化种子。在提供盐值前执行任何RCP指令将触发RCP故障。在栈金丝雀值中使用随机数据，令基于返回导向编程的栈负载难以在多次启动间重用。

图13展示了RCP硬件的数据流级概览。RCP结构为一个叠加于Cortex-M33执行流水线上的两阶段流水线。该模块通过64位输入总线（CPWDATA）和32位输出总线（CPRDATA）与核心交换数据。Cortex-M33可通过CPWDATA总线在一个周期内向协处理器发出两次寄存器读取。RCP利用此吞吐量支持部分断言指令，如 `rcp_iequal`，当两个Arm寄存器的32位值不一致时，该指令会引发故障。

图13中8位的 **tag** 值是由指令的 **CRn** 和 **CRm** 字段编码的8位立即数。

这些8位值用于唯一标识用于金丝雀值生成的函数，以防止函数之间的堆栈帧被互换。它们还为 **rcp\_count\_set** 和 **rcp\_count\_check** 指令提供8位计数器值。通过使用CRn和CRm字段对标签进行编码，使得RCP指令序列更加紧凑，避免了将这些小常量具体化到寄存器并通过CPWDATA传递而需额外指令的情况。

这也增强了标签值对故障注入攻击的抵抗力，因为指令操作码字段在周期中比通过CPWDATA传递的寄存器值更早可用。

RCP指令亦可在非安全状态下执行，但为防止非安全代码触发RCP故障或观察盐寄存器的值，存在某些差异。此举支持非安全软件执行包含RCP指令的共享只读存储器例程，但不允许从非安全上下文探查RCP的内部状态。第3.6.3.2节详述了非安全执行支持的更多细节及其设计原理。

为保证图13的清晰度，省略了部分细节，如用于伪随机指令延迟的延迟计数器，以及在非安全执行环境下抑制故障的逻辑。此行为在以下章节中有详尽说明。

### 3.6.3.1. 盐寄存器

每个RCP实例均配备一个64位盐寄存器，作为堆栈保护值和随机指令延迟的种子。该寄存器预期在启动过程早期以随机值进行初始化：RP2350只读存储器启动程序使用真随机数发生器生成盐值。

盐寄存器初始处于无效状态。该状态仅允许以下操作：

- 通过 **rcp\_canary\_status** 检查盐寄存器的有效状态
- 通过 **rcp\_salt\_core0** 或 **rcp\_salt\_core1** 写入盐值，向该核的盐寄存器写入64位数值，并将其状态置为有效

当盐寄存器处于无效状态时，执行除上述操作之外的任何RCP指令均会无条件触发RCP故障。由于RP2350只读存储器启动程序中包含大量RCP指令，这使得通过故障注入跳过RCP初始化变得极为困难。

同样，试图向已有效的RCP盐寄存器写入数据会触发RCP故障。没有理由对RCP盐寄存器进行二次初始化，因此此情况被视为表明控制流完整性丧失的异常。

核心0的协处理器端口写入两个核心RCP实例的盐寄存器，以简化多核在早期启动阶段的交互。在RP2350引导只读存储器中，核心1首先将其MPU执行权限锁定在包含等待启动代码的只读存储器小区域内，然后在核心0清理启动内存、执行必要的硬件初始化并生成RCP盐后，轮询其RCP盐的有效状态。

当核心0切换为RISC-V架构且核心1采用Arm架构时，核心1的盐寄存器会被强制标记为有效，以允许核心1执行只读存储器中的代码。由于只有在禁用安全启动时才启用RISC-V核心，此操作不会影响安全启动；将核心0设为RISC-V的能力已意味着对安全启动的破坏。

### 3.6.3.2. 非安全访问

设置Cortex-M33NSACR寄存器的第7位允许非安全代码设置 **CPACR\_NS** 的第7位，从而使非安全访问RCP成为可能。非安全RCP访问对于执行包含RCP指令的共享安全/非安全例程尤为重要。例如，RP2350只读存储器中的memcpy实现由主启动路径中的安全代码与USB引导程序等非安全代码共享。

鉴于RCP故障对系统上运行的所有软件均致命，非安全代码不得任意触发RCP故障。同样，若非安全代码能读取RCP盐寄存器，将更易构造出能控制安全执行且不触发RCP故障的堆栈载荷。因此，RCP对非安全访问的处理方式区别于安全访问：

- 将读取数据屏蔽为全零

- 忽略写入数据：任何可能引发数据相关RCP错误的指令均视为无操作
- 对无效指令报告协处理器错误，而非RCP错误，处理器将其映射为非安全的UNDEFINSTR UsageFault
- 跳过伪随机指令延迟：假设Cortex-M33能够以每周期一条指令的速度发出指令，所有RCP指令均在一个周期内执行

伪随机指令延迟的缺失使非安全代码更难提取用于向安全执行RCP指令添加延迟的种子值。

### 3.6.3.3 指令验证

RCP对所有以协处理器7为目标的协处理器指令应用以下规则：

- `Opc1`字段中1的数量加上指令奇偶校验位，必须为偶数。
  - 对于`mcr`、`mrc`和`cdp`指令，`Opc2`字段的第0位编码奇偶校验位。
  - 对于`mrr`，`CRm`字段的第3位编码奇偶校验位。
- 该指令不得为`mrc`（64位协处理器到内核指令）。
- 对于`mcr`指令（32位内核到协处理器）：
  - `Opc1`字段的值必须在0至6之间。
  - 如果没有8位标签（即非`rcp_canary_check`、`rcp_count_check`或`rcp_count_set`），则`CRn`和`CRm`操作码字段必须全部为零。
- 对于`mrc`指令（32位协处理器到内核）：
  - `Opc1`字段的值必须在0至2之间。
  - 对于除`rcp_canary_get`和`rcp_canary_check`以外的指令，`CRn`和`CRm`操作码字段必须全部为零。
- 对于`mrr`指令（64位核心到协处理器）：
  - `Opc1`字段的值必须在0至8之间。
  - 对于`rcp_salt_core*`指令，`CRm`字段的第2至0位必须为0或1（分别称为`rcp_salt_core0`和`rcp_salt_core1`）。
  - 对于所有其他`mrr`指令，`CRm`字段的第2至0位必须为0。

上述描述中的`Opc1`、`Opc2`、`CRm`和`CRn`术语，指的是Arm T32架构中协处理器指令的标准编码字段。有关编码和汇编语法的完整细节，请参阅Armv8-M架构参考手册。

任何针对协处理器7且未通过这些验证规则的协处理器指令，都会导致以下两种结果之一，具体取决于指令执行时所在的全域：

- 无效指令的安全执行将立即导致无条件的RCP故障。RCP会断言核心的不可屏蔽中断信号，任何后续的RCP指令将使协处理器端口无限期阻塞。该状态将持续，直至核心接收到热复位。这也会在其他核心触发RCP故障：更多信息详见第3.6.3.4节。
- 非安全状态下执行无效指令将在操作码阶段协处理器接口返回错误，该错误被核心解释为非安全UNDEFINSTR UsageFault。有关该Armv8-M特定故障的完整描述，详见Armv8-M架构参考手册。

### 3.6.3.4 跨核心触发

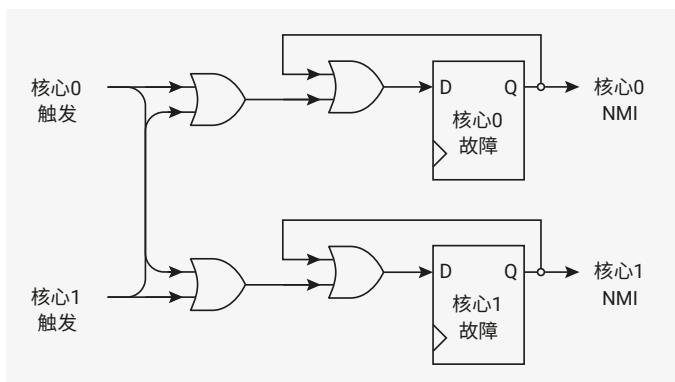
RCP故障表明软件环境的整体性已被破坏。虽故障可能起于

单个处理器，所有共享相同受信内存的处理器若继续执行，均可能表现出不可预测的行为，因为：

- 导致某个处理器以可检测方式错误执行的物理情况，例如低电压供应，可能导致其他处理器以未被检测的方式错误执行。
- 触发RCP故障的处理器可能已损坏共享且受信任的内存内容，从而干扰其他处理器的运行（例如破坏其他核心的堆栈）。

因此，一个核心上的RCP故障也会引发其他核心上的RCP故障。由于RP2350具有两个核心，核心0上的RCP故障必然触发核心1上的故障，核心1上的RCP故障同样必然触发核心0上的故障。

图14。在一个核心上触发RCP故障也会引发另一个核心上的故障。触发信号累积到故障寄存器，故障寄存器保持设定状态直至核心复位。当故障寄存器被设定时，非屏蔽中断（NMI）



每个核心本地对来自另一核心的触发信号进行逻辑或运算。左侧两个逻辑或门的输出在逻辑上等价，但这两个门保持在核心本地，以尽量减少将核心自身故障触发信号传递至自身故障寄存器的延迟。

### 3.6.3.5. 堆栈金丝雀值

金丝雀值是在函数入口写入堆栈，并在函数退出时验证，以确保：

- 退出与入口状态匹配（即从后门退出时，必须是从前门进入）
- 在函数执行过程中，堆栈未被完全覆盖

此机制有助于防范两类攻击：

- 故障注入：任何导致程序计数器受损或引发异常间接跳转的物理故障，极可能使处理器执行与函数前导代码不匹配的函数尾声。函数中任何中途的跳转最终很可能到达尾声代码段。
- 基于返回的编程（Return-oriented programming）：故意破坏栈结构可通过执行任意操作的函数尾部序列重定向控制流。利用对栈缓冲区操作缺失边界检查可能导致栈的破坏。随机生成的金丝雀值使构造此类栈载荷变得困难。

针对基于返回的编程的缓解措施在只读存储器启动代码（bootrom）中尤为重要，因为启动代码暴露的 API 表面在运行时映射于已知位置（其物理地址始终映射于 `0x00000000`）。

这提供了类似于 C 标准库的知名利用表面。

RCP 支持两条与金丝雀相关的指令，用于管理金丝雀值：

- `rcc_canary_get` 根据盐寄存器为 8 位标签生成一个 32 位值
- `rcc_canary_check` 验证给定 8 位标签的 32 位值，若该值与相同标签通过 `rcc_canary_get` 生成的值不符，将触发 RCP 故障。

32位看门狗值如下：

- 位 `7:0`：全零
- 位 `15:8`：盐的位 `7:0` 与（盐的位 `31:24` 与 8 位标签按位与）的异或

- 位 23:16：盐的位 15:8与（盐的位 39:32与8位标签按位非的按位与）的异或
- 位 31:24：盐的位 23:16与8位标签的异或

以下代码演示了如何在C语言中计算32位看门狗值：

```
uint32_t canary_value(uint64_t salt, uint8_t tag) {
 uint32_t tag_expanded =
 (uint32_t)tag |
 ((uint32_t)~tag << 8) |
 ((uint32_t)tag << 16);
 tag_expanded &= (0xff0000u | ((salt >> 24) & 0x00ffffu));
 uint32_t result24 = tag_expanded ^ salt;
 return result24 << 8;
}
```

该金丝雀值选择的原则如下：

- 不同标签保证产生不同的金丝雀值。
- 对于任意两个不同的标签，每个标签至少依赖一个该标签独有的盐位（因此即使已知一个值，也难以计算不同标签的金丝雀值）。
- 栈上的以空字符结尾的字符串操作在读取或写入金丝雀值之前即终止。

每个函数应使用不同的金丝雀标签，以防止一个函数的栈帧被另一个函数的尾部错误重用。避免将金丝雀值用于堆栈金丝雀以外的其他用途。

RP2350 只读存储器启动程序使用范围为 0x40 至 0xbf 的8位标签。其余标签可自由用于用户代码。

### 3.6.3.6. 伪随机指令延迟

默认情况下，所有RCP指令执行时均带有0到127周期的伪随机延迟。这些延迟使外部观察者难以精确地将故障注入事件与RCP指令或其保护的关键代码路径时间对齐。

#### 注意

在某些使用情况下，RCP延迟可能暴露出侧信道，从而推断处理器状态。详见RP2350-E26。

设置指令首半字的第 12位仅禁用该指令的伪随机延迟。该指令在单个周期内执行，前提是Cortex-M33未因其他微架构约束插入停顿周期。要设置此位，请使用任何协处理器指令的 \*2变体进行汇编（例如使用 mrc2而非 mrc）。在非安全状态下，RCP指令始终无延迟执行。

RCP通过在操作码阶段（如图13流水线图所示）暂停协处理器操作码接口实现指令执行延迟。由于中断，Cortex-M33可能会选择放弃已暂停的协处理器指令。当此情况发生时，延迟计数器将继续倒计时，等待延迟周期届满。若 Cortex-M33 在延迟计数器仍在运行期间（无论是在中断中，还是返回至被中断的 RCP 指令后）发出另一条 RCP 指令，则该指令将在现有倒计时完成后执行。然而，若被废弃指令的延迟计数器在下一条 RCP 指令执行前已到期，则下一条指令将采样伪随机延迟计数，并启动新一轮倒计时。

伪随机延迟序列是盐寄存器第 63 至 40 位的函数。因此，前提是每次启动向盐寄存器写入不同的 64 位值，每次启动的延迟模式均独一无二。

用于延迟的伪随机数生成器（PRNG）在盐寄存器第 63 至 40 位中实现多个小型线性反馈移位寄存器（LFSRs），并返回该 24 位状态的非线性函数。24位状态的LFSR反馈函数为：

- 位 **23:20**: 4位LFSR，抽头为 **0xc**
- 位 **19:15**: 5位LFSR，抽头为 **0x14**
- 位 **14:8**: 7位LFSR，抽头为 **0x60**
- 位 **7:0**: 8位LFSR，抽头为 **0xb4**

LFSRs通过对（状态 AND 抽头）执行异或合并，然后将结果移入最低有效位以实现，每次状态更新时均执行此操作。当LFSR状态全为零时，将一位1移入最低有效位。每次生成随机数时，LFSR状态都会前进：该情况发生于执行带伪随机延迟的指令或执行 `rcc_random_byte` 指令时。

伪随机输出的每一位为24位状态中6个位的异或结果，且与状态中另外3个位的多数投票（三取二）结果异或：

| 输出位 | 异或抽头 |    |    |    |    |    |    | 三数多数抽头 |    |  |
|-----|------|----|----|----|----|----|----|--------|----|--|
| 7   | 7    | 17 | 6  | 16 | 13 | 8  | 9  | 12     | 21 |  |
| 6   | 14   | 21 | 19 | 6  | 16 | 13 | 4  | 14     | 6  |  |
| 5   | 7    | 5  | 2  | 18 | 11 | 1  | 18 | 14     | 7  |  |
| 4   | 4    | 19 | 17 | 0  | 18 | 7  | 18 | 11     | 3  |  |
| 3   | 23   | 12 | 7  | 16 | 14 | 5  | 17 | 3      | 15 |  |
| 2   | 15   | 13 | 20 | 21 | 8  | 12 | 7  | 22     | 9  |  |
| 1   | 4    | 16 | 11 | 18 | 9  | 6  | 14 | 21     | 16 |  |
| 0   | 11   | 3  | 4  | 19 | 10 | 14 | 1  | 2      | 9  |  |

该函数的位 **6:0** 用于伪随机指令延迟，产生0至127周期范围内的延迟。

此延迟是在执行协处理器指令的一个周期基准成本之外额外施加的。完整的8位结果可通过 `rcc_random_byte` 指令获得。

这是一个简单的伪随机数生成器，使从少量观测中恢复初始24位状态变得困难。它通过将观测规模远小于状态规模，并使用非线性组合函数来输出，从而实现此目的。其存在多处统计异常，故不适合用于通用随机数生成（更别提其状态规模较小）。对于高质量随机数生成，建议直接使用系统真实随机数生成器（TRNG），或使用由TRNG播种的大状态高质量软件伪随机数生成器。

请注意，用于为延迟伪随机数生成器（PRNG）设定种子的盐值的最高24位（MSBs）与用于生成栈保护值的最低40位（LSBs）不重叠。因此，外部测量随机延迟无法获取任何关于栈保护值的信息。

### 3.6.3.7. 指令列表

Cortex-M33处理器通过 `mcr`、`mcr2`、`mrc` 及 `cdp` 指令访问RCP。Armv8-M架构参考手册详细描述了这些指令与处理器架构状态的复杂关系，但从协处理器视角来看：

- `mcr` 将一个32位值从单个Arm整数寄存器写入协处理器
- `mcr2` 将一个64位值从一对Arm整数寄存器写入协处理器
- `mrc` 从协处理器读取一个32位值，写入单个Arm整数寄存器或处理器状态标志
- `cdp` 执行某些内部协处理器操作，且不与处理器交换数据

对于每条 `mcr`、`mcr2`、`mrc` 和 `cdp` 指令，RCP同样支持对应的 `mcr2`、`mcr2`、`mrc2` 及 `cdp2` 操作码变体。这些操作码仅在第 12 位上存在差异。普通版本的执行过程中会有最多127个周期的伪随机延迟，

而带有 `2` 后缀的版本则无此延迟。

大多数RCP指令表现为硬件校验断言形式。下述指令列表中“断言”为”的表达意指：若某断言条件不成立，协处理器将触发RCP故障。

### 3.6.3.7.1. 初始化

#### `rcp_salt_core0`

断言核心0的盐值寄存器当前处于无效状态。写入一个64位数值，并将其标记为有效。

操作码：

```
mcrr p7, #8, Rt, Rt2, c0
```

`Rt` 是盐值的32位最低有效位，`Rt2`是32位最高有效位。

#### `rcp_salt_core1`

断言核心1的盐寄存器当前无效。写入一个64位数值，并将其标记为有效。

操作码：

```
mcrr p7, #8, Rt, Rt2, c1
```

#### `rcp_canary_status`

返回一个 `true`或 `false`的位模式（分别为 `0xa500a500`或 `0x00c300c3`），用于指示该核心的盐寄存器是否已初始化。

操作码：

```
mrc p7, #1, Rt, c0, c0, #0
```

仅当盐寄存器有效时，调用时传入 `Rt = 0xf`会设置Arm的 `N`和 `C`标志位。

如果盐尚未初始化，除初始化盐或检查canary状态外的任何操作都会触发RCP故障。

该操作码在核心0用于跳过RCP初始化序列（当在调试器控制下未复位重新进入bootrom时），核心1用于等待其RCP 盐值初始化完成。

### 3.6.3.7.2. 金丝雀

#### `rcp_canary_get`

获取一个32位金丝雀值，该值基于盐寄存器及由两个4位协处理器寄存器编号 `CRn`和 `CRm`编码的8位标签。`CRn`包含四个最高有效位，`CRm`包含四个最低有效位。

操作码：

```
mrc p7, #0, Rt, CRn, CRm, #1
```

第3.6.3.5节规定了该指令返回的32位值，但应将其视为不透明值，由 `rcp_canary_check` 使用。

**rcp\_canary\_check**

断言某值与指定相同8位标签的 `rcp_canary_get` 结果一致。标签由两个4位协处理器寄存器编号 CRn 和 CRm 编码。 CRn 包含四个最高有效位， CRm 包含四个最低有效位。

操作码：

```
mcr p7, #0, Rt, CRn, CRm, #1
```

**3.6.3.7.3. 布尔验证**

RCP 将 `0xa500a500` 定义为 32 位布尔值的 `true` 值，将 `0x00c300c3` 定义为 `false` 值。所有其他位模式均为无效，若被任何 RCP 布尔指令使用，将触发 RCP 错误。这些值之所以被选用，是因为它们在 Armv8-M Main 中是有效的立即数。

此机制提供有限的运行时类型检查，以确保布尔值仅在布尔上下文中使用。RP2350 启动只读存储器偶尔使用冗余操作生成布尔值，若两个冗余操作的返回值不一致（例如在 OTP 中检查启动标志时），将产生无效的位模式。

**rcp\_bvalid**

断言 Rt 为有效布尔值（`0xa500a500` 或 `0x00c300c3`）。

操作码：

```
mcr p7, #1, Rt, c0, c0, #0
```

**rcp\_btrue**

断言 Rt 为 `true`（`0xa500a500`）。

操作码：

```
mcr p7, #2, Rt, c0, c0, #0
```

**rcp\_bfalse**

断言 Rt 为 `false`（`0x00c300c3`）。

操作码：

```
mcr p7, #3, Rt, c0, c0, #1
```

**rcp\_b2valid**

断言 Rt 和 Rt2 均为有效布尔值。

操作码：

```
mcrr p7, #0, Rt, Rt2, c8
```

**rcp\_b2and**

断言 Rt 和 Rt2 均为真。

操作码：

```
mcrr p7, #1, Rt, Rt2, c0
```

**rcp\_b2or**

断言 Rt 和 Rt2 均有效，且至少一个为真。

```
mcrr p7, #2, Rt, Rt2, c0
```

**rcp\_bxorvalid**

断言 Rt 异或 Rt2 为有效布尔值。XOR 掩码通常为固定比特模式，用于验证布尔值的来源，例如关键函数返回值。

操作码：

```
mcrr p7, #3, Rt, Rt2, c8
```

**rcp\_bxortrue**

断言 Rt XOR Rt2 为真。

操作码：

```
mcrr p7, #4, Rt, Rt2, c0
```

**rcp\_bxorfalse**

断言 Rt XOR Rt2 为假。

操作码：

```
mcrr p7, #5, Rt, Rt2, c8
```

**3.6.3.7.4. 整数验证****rcp\_ivalid**

断言 Rt XOR Rt2 等于 0x96009600。此方法用于验证冗余存储在两个内存字中的32位整数。XOR 差异确保两条并行的整数运算链未被干扰。

操作码：

```
mcr p7, #6, Rt, Rt2, c8
```

**rcp\_iequal**

断言 **Rt** 等于 **Rt2**。对于值得在硬件中进行检测的一般软件断言，此项极为有用。

操作码：

```
mcr p7, #7, Rt, Rt2, c0
```

**3.6.3.7.5. 随机****rcp\_random\_byte**

返回从64位盐值的高24位生成的随机8位数值。结果的第**31**至第**8**位均为零。

操作码：

```
mrc p7, #2, Rt, c0, c0, #0
```

该伪随机数生成器与用于随机延迟值的相同。此处主要用于调试目的，不应用于一般软件随机数生成，因为24位的状态空间无法满足对随机数质量与不可预测性有较高要求的应用场景。

该指令不会产生执行延迟。Cortex-M33发出协处理器访问请求后，必定在一个周期内完成。

**3.6.3.7.6. 序列计数检查**

这些指令用于断言一系列操作按正确顺序执行。计数在序列开始时初始化为8位值，随后反复检查，每次检查时递增。

如果8位校验值与当前计数器值不符，协处理器将触发RCP故障。

**rcp\_count\_set**

向RCP序列计数器写入8位计数值。使用两个4位协处理器编号编码8位值：**CRn**提供高位，**CRm**提供低位。

操作码：

```
mcr p7, #4, r0, CRn, CRm, #0
```

**rcp\_count\_check**

断言8位计数值与RCP序列计数器当前值相符。计数器递增1，计数达到 **0xff** 后回绕至 **0x00**。使用两个4位协处理器编号编码8位计数值：**CRn**提供高位，**CRm**提供低位。

操作码：

```
mcr p7, #5, r0, CRn, CRm, #1
```

### 3.6.3.7.7. 紧急情况

#### `rcp_panic`

永久阻塞协处理器端口。如果处理器放弃协处理器访问，则断言非屏蔽中断（NMI）并持续阻塞协处理器端口。并立即在其他核心上引发RCP故障。

操作码：

```
cdp p7, #0, c0, c0, c0, #1
```

软件在检测到当前程序继续执行存在安全隐患时，会执行 `rcp_panic` 指令。RCP的响应是永久阻塞处理器对CDP的访问，这应导致处理器停止获取和执行指令。

当处理器被中断时，允许其放弃阻塞的协处理器指令，但这可能导致继续在不安全状态下执行。RCP通过断言非屏蔽中断响应被放弃的传输，抢占触发协处理器访问放弃的中断处理程序。此时应迅速遇到另一个RCP指令，并再次阻塞处理器，且此次不允许中断。

Panic采用这种方式指定，而非通过门控处理器时钟，使调试器在发生panic后仍能清晰地附加至处理器。

## 3.6.4. 浮点运算单元

RP2350上的Cortex-M33内核配备标准的Arm单精度浮点运算单元（FPU）。

协处理器端口 [10](#) 和 [11](#) 可访问FPU。

Arm浮点扩展详见Armv8-M架构参考手册。

使用SDK构建的应用程序默认自动调用FPU。例如，在C语言中，对 `float` 数据类型的计算自动使用标准FPU，而对 `double` 数据类型的计算自动使用RP2350双精度协处理器（见第3.6.2节）。

## 3.7. Cortex-M33 处理器

### Arm文档

下列内容大部分摘自Cortex-M33技术参考手册，经授权使用。

Arm Cortex-M33处理器是一款逻辑门数低、高能效的处理器，适用于微控制器及嵌入式应用。该处理器基于Armv8-M架构，主要应用于对安全性有重要要求的环境。

### ⓘ 注意

有关Arm Cortex-M33处理器的完整详情，请参阅技术参考手册。

## 3.7.1. 特性

Arm Cortex-M33处理器提供以下功能与优势：

- 顺序发射流水线
- Thumb-2技术；如需更多信息，请参阅Arm v8-M架构参考手册。
- 小端数据访问
- 与处理器紧密集成的嵌套向量中断控制器（NVIC）
- 支持单精度浮点运算的浮点单元（FPU）
- 支持可异常续发的指令，如LDM、LDMDB、STM、STMDB、PUSH、POP、VLDM、VSTM、VPUSH和VPOP
- 一种低成本调试解决方案，具备实现以下功能的能力：
  - 断点
  - 观察点
  - 跟踪
  - 系统性能分析
  - 通过仪器追踪宏单元（ITM）支持printf()风格的调试
- 支持嵌入式追踪宏单元（ETM）指令追踪选项；如需更多信息，请参阅Arm CoreSight ETM-M33技术参考手册
- 
- 用于外部硬件加速器的协处理器接口
- 低功耗特性，包括架构时钟门控、睡眠模式及具备唤醒中断控制器（WIC）的电源感知系统
- 包括内存保护与安全属性的内存系统

## 3.7.2. 配置

RP2350 中每个 Arm Cortex-M33 处理器均配置以下特性：

- **FPU：单精度浮点运算单元**
- DSP：数字信号处理扩展
- **SECEXT：安全扩展**
- CPIF：协处理器接口
- MPU\_NS：8 个非安全 MPU 区域
- MPU\_S：8 个安全 MPU 区域
- SAU：8 个安全属性单元（SAU）区域
- IRQ：52 个外部中断
- IRQLVL：4 位异常优先级
- DBGLVL：完整调试集，包括 4 个观察点、8 个断点比较器及调试监视器

- ITM：数据观察追踪（DWT）与 ITM 追踪
- ETM：指令追踪（ETM）
- MTB：无 MTB 追踪
- WIC：唤醒中断控制器
- WICLINES: 55：所有外部中断及3个内部事件：NMI、RVEV、调试
- CTI：交叉触发接口
- RAR：上电时复位所有寄存器
- UNCROSS\_I\_D：修改内部地址映射
- SBIST：无SBIST功能
- 未使用CDE模块
- CDERTLID：多Cortex-M33系统的RTL ID：16

架构时钟门控允许处理器核心通过禁用核心部分时钟，支持SLEEP和DEEPSLEEP低功耗状态。不支持电源门控。

每个Cortex-M33核心拥有独立的中断控制器，可以根据需要单独屏蔽中断源。相同的中断会路由到两个Cortex-M33核心。

处理器支持以下接口：

- 代码AHB（C-AHB）接口
- 系统AHB（S-AHB）接口
- 外部PPB（EPPB）APB接口
- 调试AHB（D-AHB）接口

处理器实现了以下可选接口：

- Arm TrustZone 技术，采用支持安全与非安全状态的 Armv8-M 安全扩展
- 内存保护单元（MPU），可配置以保护特定内存区域
- 具备支持单精度算术运算的浮点运算功能
- 支持 ETM 跟踪功能

### 3.7.2.1. Raspberry Pi 的修改内容

#### 3.7.2.1.1. UNCROSS\_I\_D

原始 Cortex-M33 处理器设计将以下操作路由至代码端口或系统端口：

- 指令取指
- 加载/存储
- 调试器访问

地址低于 `0x2` 的访问 `0000000` 路由至代码端口。所有其他访问路由至系统端口。

该路由策略可能引发内部总线矩阵和主系统 AHB5 交叉开关的争用。

Cortex-M33 技术参考手册对此策略有详细描述。

在 RP2350 中，Raspberry Pi 对 Cortex-M33 总线矩阵进行了修改，具体为：

- 将所有指令取指操作路由至代码端口

- 将所有加载/存储和调试器访问路由至系统端口

此举消除内部冲突，并在某些软件使用场景中提升性能，例如当从单一统一的SRAM池分配代码与数据时。

在第3.7.2节中，我们将该功能称为UNCROSS\_I\_D。

Cortex-M33处理器未作其他修改。

### 注意

本数据手册可能将Cortex-M33的代码端口与系统端口分别称为指令端口与数据端口（I与D），以反映对内核集成总线矩阵的该项修改。

## 3.7.2.2 接口

处理器具备多种外部接口：

### 代码与系统 AHB 接口

采用哈佛AHB总线架构，支持排他性事务与安全状态。

### 系统 AHB 接口

系统 AHB（S-AHB）接口用于任何对Armv8-M存储映射的SRAM、外设、外部RAM、外部设备或Vendor\_SYS区域的指令提取和数据访问。

### 代码 AHB 接口

代码 AHB（C-AHB）接口用于对 Armv8-M 存储器映射中的代码区域进行任何指令取指和数据访问。

### 外部私有外设总线

外部 PPB (EPPB) APB 接口使系统能够访问连接至处理器的兼容 CoreSight 调试和追踪组件。

### 安全归因接口

处理器设有连接至外部实现定义归因单元（IDAU）的接口，允许系统基于地址设置安全属性。

### ATB 接口

ATB 接口用于输出调试追踪数据。ATB 接口兼容 CoreSight 架构。更多信息请参阅 Arm CoreSight 架构规范 v2.0。指令 ATB 接口由 ETM 使用，仪表 ATB 接口由仪表追踪宏单元（ITM）使用。

### 微追踪缓冲区接口

微追踪缓冲区（MTB）的 AHB 从接口和 SRAM 接口用于 CoreSight 微追踪缓冲区。

### 协处理器接口

协处理器接口专为紧密耦合的外部加速硬件设计。

### 调试 AHB 接口

调试 AHB（D-AHB）从属接口允许调试器访问寄存器、内存和外设。D-AHB 接口提供对处理器及完整内存映射的调试访问。

### 交叉触发接口

处理器包含一个交叉触发接口（CTI）单元，该接口适用于通过交叉触发矩阵（CTM）连接外部 CoreSight 组件。

### 电源控制接口

处理器支持多个内部电源域，可通过连接至系统中电源管理单元（PMU）的 Q 通道接口启用或禁用。

### 3.7.2.3 安全归属与内存保护

Cortex-M33 处理器支持 Armv8-M 受保护内存系统架构（PMSA），该架构利用多个软件可控区域实现可编程内存保护。RP2350 支持 8 个可编程区域。

PMSA 允许特权软件为内存区域分配访问权限。当非特权软件尝试访问该区域时，将触发故障异常。PMSA 包含故障状态寄存器，允许异常处理程序确定故障来源，采取纠正措施并通知系统。此机制减少了错误编写的应用程序代码可能造成的潜在影响。

Cortex-M33 处理器还支持根据 Armv8-M 安全扩展，将内存区域定义为安全或非安全状态，从而防止不适当安全级别的访问。

### 3.7.2.4. 浮点单元 (FPU)

FPU 提供以下功能：

- 针对单精度（C 语言 float 类型）数据处理操作的指令
- 针对双精度（C 语言 double 类型）加载与存储操作的指令
- 用于提高精度的乘加合并指令（Fused MAC）
- 支持硬件实现的转换、加法、减法、乘法、累加、除法及平方根操作
- 对非正规数及所有 IEEE Standard 754-2008 舍入模式的硬件支持
- 三十二个 32 位单精度寄存器或十六个 64 位双精度寄存器
- 惰性浮点上下文保存

#### 3.7.2.4.1. 惰性浮点上下文保存

该 FPU 功能延迟自动堆栈保存浮点状态，直至中断服务例程（ISR）尝试执行浮点指令。此举减少了进入 ISR 的时延，并避免了对不使用浮点的 ISR 进行浮点上下文保存。

### 3.7.2.5. NVIC

嵌套向量中断控制器（NVIC）负责对外部中断信号进行优先级排序。软件能够设置各中断的优先级。NVIC 与 Cortex-M3 3 处理器核心紧密耦合，提供低延迟的中断处理及对后续中断的高效响应。

#### ① 注意

“嵌套”意指中断本身可以被更高优先级的中断打断。“向量化”意指硬件将每个中断分派至矢量表中指定的独立处理程序例程。有关嵌套和矢量行为的详细信息，请参阅 Armv8-M 架构参考手册。

所有 NVIC 寄存器仅能通过字传输访问。任何单独读取或写入半字或字节的尝试行为均为不可预测。

NVIC 寄存器始终采用小端序存储。

嵌套矢量中断控制器（NVIC）与内核紧密集成，以实现低延迟的中断处理。

NVIC 的功能包括：

- 外部中断，支持1至480个中断，可通过连续或非连续映射进行配置。该配置由具体实现决定。
- 可配置中断优先级级别，范围从8至256。该配置由具体实现决定。
- 支持中断的动态优先级重新排序。
- 优先级分组功能。该功能允许选择抢占式优先级和非抢占式优先级。
- 支持尾部链式和延迟到达中断。该功能使得连续中断处理无需中断间的状态保存与恢复开销。
- 支持 Armv8-M 安全扩展。安全中断的优先级可高于任何非安全中断。

### 3.7.2.6. 交叉触发接口单元（CTI）

CTI使调试逻辑、MTB和ETM能够相互交互，并与其他CoreSight™组件协同工作。

### 3.7.2.7. ETM

ETM仅提供指令执行追踪功能。

### 3.7.2.8. MTB

MTB为Cortex-M33处理器提供一种简易且低成本的执行跟踪解决方案。

跟踪数据写入SRAM接口，并可通过处理器上的专用AHB从属接口（M-AHB）提取。MTB可通过PPB区域的内存映射寄存器控制，或由DWT生成的事件或CTI触发。

更多详情请参阅Arm CoreSight MTB-M33技术参考手册。

### 3.7.2.9. 调试与跟踪

调试与跟踪组件包括可配置断点单元（BPU），用于实现断点；以及可配置数据观察点与跟踪单元（DWT），用于实现观察点、数据跟踪和系统性能分析。

其他调试和追踪组件包括：

- 支持 `printf()` 风格调试的ITM，基于仪表追踪
- 适用的接口包括：
  - 通过追踪端口接口单元（TPIU），利用4位DDR输出作为GPIO功能（参见第3.5.7节），将片上数据传递至追踪端口分析仪（TPA）
  - 只读存储器（ROM）表，允许调试器确定Cortex-M33中已实现的组件处理器
  - 调试器可访问系统中的所有内存和寄存器，包括内存映射设备，在核心停止时访问内部核心寄存器，且即便复位信号被断言，依然可访问调试控制寄存器

### 3.7.3. 合规性

该处理器符合或实现了相关Arm架构标准和协议，以及相关外部标准。

### 3.7.3.1 Arm架构

该处理器符合以下规范：

- Armv8-M主扩展
- Armv8-M安全扩展
- Armv8-M保护内存系统架构（PMSA）
- Armv8-M 浮点扩展
- Armv8-M 数字信号处理（DSP）扩展
- Armv8-M 调试扩展
- Armv8-M 闪存补丁断点（FPB）架构版本 2.0

### 3.7.3.2 总线架构

处理器提供符合 AMBA 5 AHB5 协议的外部接口。处理器还采用 APB4 协议及 AMBA 4 ATB 协议中的 ATBv1.1，为 CoreSight 及其他调试组件实现接口。

更多信息，请参阅：

- [Arm AMBA 5 AHB 协议规范](#)
- [AMBA APB 协议版本 2.0 规范](#)
- [Arm AMBA 4 ATB 协议规范，含 ATBv1.0 与 ATBv1.1](#)

处理器还提供 Q-Channel 接口。更多信息，请参阅 AMBA 低功耗接口规范。

### 3.7.3.3 调试

处理器的调试功能实现了 Arm 调试接口架构。更多信息，请参阅 Arm 调试接口架构规范 ADIv5.0 至 ADIv5.2。

### 3.7.3.4. 嵌入式跟踪宏单元

处理器的跟踪功能实现了 Arm 嵌入式跟踪宏单元（ETM）v4.2 架构。

如需更多信息，请参见 Arm CoreSight ETM-M33 技术参考手册。

### 3.7.3.5. 浮点单元

Cortex-M33 处理器配备的浮点单元（FPU）支持由 Armv8-M 架构中 FPv5 定义的单精度算术运算。该 FPU 提供符合 ANSI/IEEE 标准 754-2008 —— 二进制浮点运算 IEEE 标准的浮点计算功能。

FPU 支持单精度的加、减、乘、除、乘加累积及平方根运算。  
此外，还支持固定点与浮点数据格式之间的转换及浮点常量指令。

FPU 包含一个扩展寄存器文件，含 32 个单精度寄存器。

这些寄存器可视为：

- 三十二个 32 位单字寄存器, [S0-S31](#)
- 十六個 64 位双字寄存器, [D0-D15](#)
- 这些视图中寄存器的组合

### 3.7.3.5.1. FPU 模式

FPU 提供完全兼容、零舍入 (Flush-to-Zero) 和默认 NaN 模式的操作。在完全兼容模式下，FPU 通过硬件根据 IEEE 754 标准处理所有操作。

操作模式由浮点状态和控制寄存器 ([FPSCR](#)) 控制。

设置 [FPSCR.FZ](#) 位即可启用零舍入 (Flush-to-Zero, FZ) 模式。在 FZ 模式下，FPU 将所有算术操作的非规格化输入操作数视为零。由零操作数引发的异常会被适当报告。[VABS](#)、[VNEG](#) 和 [VMOV](#) 不被视为算术操作，因此不受 FZ 模式影响。当操作产生 *tiny* 结果时（如 IEEE 754 标准规定，目标精度的数值在舍入前小于最小正规值），FZ 模式将结果替换为零。

[FPSCR.IDC](#) 位指示输入刷新事件的发生。

[FPSCR.UFC](#) 位指示结果刷新事件的发生。

设置 [FPSCR.DN](#) 位以启用默认 NaN (DN) 模式。在 NaN 模式下，任何涉及输入 NaN 或产生 NaN 结果的算术数据处理操作，其结果均返回默认 NaN。除 [VABS](#)、[VNEG](#) 和 [VMOV](#) 外，所有算术操作均忽略输入 NaN 的分数位。

当 [FPSCR.DN](#) 位和 [FPSCR.FZ](#) 位均未设置时，启用全合规模式。在全合规模式下，[FPv5](#) 功能在硬件层面符合 IEEE 754 标准。

有关 FPU 和 FPSCR 的更多信息，请参见 Armv8-M 架构参考手册。

### 3.7.3.5.2 FPU 异常

FPU 根据 FPv5 架构要求，在 FPSCR 寄存器中为每条指令设置累计异常状态标志。FPU 不支持异常陷阱。

处理器具有六个输出引脚。默认情况下，它们处于断开状态。每个引脚反映一个累积异常标志的状态：

#### FPIXC

屏蔽的浮点不精确异常。

#### FPUFC

屏蔽的浮点下溢异常。

#### FPOFC

屏蔽的浮点溢出异常。

#### FPDZC

屏蔽的浮点除零异常。

#### FPIDC

屏蔽的浮点输入非规格化数异常。

#### FPIOC

无效操作。

当浮点上下文处于活动状态时，堆栈帧会扩展以容纳浮点寄存器。为减少异常进入时写入更大堆栈帧所带来的额外中断延迟，处理器支持惰性堆栈技术。这意味着处理器在堆栈上预留了浮点状态的空间，但仅在异常处理程序内执行 FPU 指令时，才将该状态信息保存至堆栈。

FP 状态的延迟保存可被更高优先级的异常中断。该异常返回后，FP 状态保存操作将重新执行。

### 3.7.3.5.3. 低功耗 FPU 操作

若 FPU 处于独立电源域，FPU 域的断电方式取决于该域是否含有状态保持逻辑。

关闭 FPU 电源的步骤如下：

- 若 FPU 域包含状态保持逻辑，通过清除 CPACR.CP10 和 CPACR.CP11 位字段以禁用 FPU。
- 若 FPU 域不包含状态保持逻辑，通过清除 CPACR.CP10 和 CPACR.CP11 位字段以禁用 FPU，并将 CPPWR.SU10 和 CPPWR.SU11 位字段均设为 1。

#### — 警告

设置 CPPWR.SU10 和 CPPWR.SU11 位字段即表示可能丢失 FPU 状态。

## 3.7.4. 程序员模型

Cortex-M33 程序员模型是 Armv8-M 主扩展架构的实现。

有关程序员模型的完整描述，请参阅 Armv8-M 架构参考手册，该手册还包含 Armv8-M Thumb 指令。此外，系统控制、MPU、NVIC、FPU、调试、DWT、ITM 和 TPIU 等功能主题中，描述了程序员模型的其他选项。

### 3.7.4.1. 操作模式与执行

Cortex-M33 处理器支持安全状态与非安全状态、安全线程和处理程序模式，并且可在 Thumb 状态或调试执行状态下运行。此外，处理器可以通过在特权模式或非特权模式下执行代码，限制或拒绝对某些资源的访问。

有关操作模式与执行的更多信息，请参阅 Armv8-M 架构参考手册。

#### 3.7.4.1.1. 安全状态

借助 Armv8-M 安全扩展，程序员模型包含两个相互正交的安全状态：安全状态和非安全状态。处理器始终复位进入安全状态。每个安全状态包含一组独立的操作模式，并支持特权级和非特权级用户访问。系统控制空间中的寄存器在安全状态与非安全状态之间采用寄存器组切换，且安全状态通过别名地址访问非安全寄存器视图。

#### 3.7.4.1.2. 操作模式

对于每个安全状态，处理器可在线程（*Thread*）或处理程序（*Handler*）模式下运行。以下条件将导致处理器进入线程模式或处理程序模式：

- 处理器在复位时进入线程模式，或在异常返回至线程模式时进入线程模式。特权级和非特权级代码均可在线程模式下执行。
- 处理器在发生异常时进入处理程序模式。在处理程序模式下，所有代码均具有特权级。

处理器可在响应异常时切换安全状态，例如，当从非安全状态捕获安全异常时，线程模式将进入安全状态的处理程序模式。处理器还可调用安全函数。

来自非安全态及安全态中的非安全函数。安全扩展包含对这些调用的要求，以防止安全数据在非安全态被访问。

### 3.7.4.1.3. 运行状态

处理器可在Thumb状态或调试状态下工作：

- Thumb状态为正常执行状态，运行16位和32位半字对齐的Thumb指令。
- 调试状态为处理器处于停止调试状态时的状态。

### 3.7.4.1.4. 特权访问与非特权用户访问

代码可以以特权或非特权身份执行。非特权执行限制资源访问，符合当前安全状态。特权执行可访问安全状态下的所有资源。异常处理程序模式始终为特权模式。线程模式可为特权或非特权模式。

### 3.7.4.2. 指令集摘要

处理器实现了Armv8-M架构中的以下指令：

- 所有基础指令
- 主扩展中的所有指令
- 安全扩展中的所有指令
- 数字信号处理扩展中的所有指令
- 浮点扩展中的所有单精度指令及双精度加载/存储指令

有关 Armv8-M 指令的更多信息，请参阅 Armv8-M 架构参考手册。

### 3.7.4.3 内存模型

处理器包含一个总线矩阵，用于在外部内存系统与内部系统控制空间（SCS）及调试组件之间，仲裁处理器核心的指令获取和内存访问。

通常优先保证处理器访问，以尽可能减少调试访问的侵入性。

系统内存映射符合 Armv8-M 主扩展规范，调试器和处理器访问共享该映射。

默认内存映射为除私有外设总线（PPB）外的所有区域提供用户和特权访问权限。

PPB 空间仅允许特权访问。

下表显示默认内存映射。该映射适用于内置 MPU 被禁用的情况。除针对 NVIC 和调试组件的区域外，所有区域的属性和权限均可通过已实现的 MPU 进行修改。

表114. 默认内存映射

| 地址范围（含端点）               | 区域   | 接口                                               |
|-------------------------|------|--------------------------------------------------|
| 0x00000000 - 0x1FFFFFFF | 代码   | 指令和数据访问。                                         |
| 0x20000000 - 0x3FFFFFFF | SRAM | 指令和数据访问。                                         |
| 0x40000000 - 0x5FFFFFFF | 外设   | 指令和数据访问。任何尝试从外设和外部设备区域执行指令的操作，均会导致 MemManage 错误。 |

| 地址范围（含端点）               | 区域         | 接口                                                                                                                                                                           |
|-------------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x60000000 - 0xFFFFFFFF | 外部 RAM     | 指令和数据访问。任何尝试从外设和外部设备区域执行指令的操作，均会导致 MemManage 错误。                                                                                                                             |
| 0xA0000000 - 0xDFFFFFFF | 外部设备       | 指令和数据访问。任何尝试从外设和外部设备区域执行指令的操作，均会导致 MemManage 错误。                                                                                                                             |
| 0xE0000000 - 0xE00FFFFF | PPB        | 保留用于系统控制和调试。不可用于异常向量表。数据访问要么在内部执行，要么在EPPB上执行。访问范围为 0xE0000000 - 0xE0043FFF 的访问由处理器内部处理。访问范围为 0xE0044000 - 至 0xE0FFFFF 的访问在处理器的EPPB接口上表现为APB事务。任何尝试从该区域执行指令的行为将导致MemManage故障。 |
| 0xE0100000 - 0xFFFFFFFF | Vendor_SYS | 部分保留用于未来处理器功能扩展。任何尝试从该区域执行指令的行为将导致MemManage故障。                                                                                                                               |

内部安全归属单元（SAU）确定与地址相关的安全级别。一些内部外设在PPB区域有内存映射寄存器，这些寄存器在安全状态和非安全状态之间分银行。

当处理器处于安全状态时，软件可访问这些寄存器的安全版本和非安全版本。非安全版本通过别名地址进行访问。

有关内存模型的更多信息，请参见 Armv8-M 架构参考手册。

### 3.7.4.3.1. 私有外设总线（PPB）

私有外设总线（PPB）内存区域提供对内部及外部处理器资源的访问。

内部 PPB 提供对以下内容的访问：

- 系统控制空间（SCS），包括内存保护单元（MPU）、安全归属单元（SAU）以及嵌套向量中断控制器（NVIC）。
- 数据监视点与跟踪单元（DWT）。
- 断点单元（BPU）。
- 嵌入式跟踪宏单元（ETM）。
- CoreSight 微跟踪缓冲区（MTB）。
- 交叉触发接口（CTI）。
- 只读存储器表（ROM table）。

外部 PPB（EPPB）提供对 PPB 内存映射中实现特定外部区域的访问。

### 3.7.4.3.2. 非对齐访问

Cortex-M33 处理器支持非对齐访问。它们会被转换为处理器上的 C-AHB 或 S-AHB 主端口的两个或多个对齐 AHB 事务。

非对齐支持仅适用于普通内存地址上的单次加载/存储指令（LDR、LDRH、STR、STRH、TBH）。加载/存储双字及加载/存储多字指令已经支持字对齐访问，但不允许其他非对齐访问，且若尝试将产生故障。设备内存中的非对齐访问禁止且会引发故障。跨越内存映射边界的非对齐访问在体系结构上为不可预测。

### ⓘ 注意

若当前安全状态的`CCR.UNALIGN_TRP`已设置，任何非对齐访问都会引发故障。

#### 3.7.4.4. 排他监视器

Cortex-M33 处理器实现了本地排他监视器。处理器内的本地监视器设计为不保存任何物理地址，而是将所有存储排他访问视为匹配先前加载排他访问的地址。这意味着已实现的独占保留粒度为整个内存地址范围。有关信号量及本地独占监视器的详细信息，请参阅Armv8-M架构参考手册。

#### 3.7.4.5. 处理器内核寄存器汇总

下表展示了处理器内核寄存器集的汇总。每个寄存器宽度均为32位。包含Armv8-M安全扩展时，部分寄存器为分组寄存器。当Cortex-M33处理器处于安全态时，可访问这些寄存器的安全视图；处于非安全态时，则访问非安全视图。

表115。处理器内核寄存器集  
汇总

| 名称                     | 描述                                                                                                                                                                                            |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>R0-R12</code>    | <code>R0-R12</code> 为用于数据操作的通用寄存器。                                                                                                                                                            |
| <code>MSP (R13)</code> | 堆栈指针（SP）对应寄存器 <code>R13</code> 。在线程模式下， <code>CONTR</code> 寄存器指示所用的堆栈指针类型，即主堆栈指针（MSP）或进程堆栈指针（PSP）。<br>Cortex-M33 处理器中存在两个 MSP 寄存器：<br><code>MSP_NS</code> 表示非安全状态， <code>MSP_S</code> 表示安全状态。 |
| <code>PSP (R13)</code> | 堆栈指针（SP）对应寄存器 <code>R13</code> 。在线程模式下， <code>CONTR</code> 寄存器指示所用的堆栈指针类型，即主堆栈指针（MSP）或进程堆栈指针（PSP）。<br>Cortex-M33 处理器中存在两个 PSP 寄存器：<br><code>PSP_NS</code> 表示非安全状态， <code>PSP_S</code> 表示安全状态。 |
| <code>MSPLIM</code>    | 栈限制寄存器分别限制 MSP 和 PSP 寄存器的最低有效地址。Cortex-M33 处理器中存在两个 MSPLIM 寄存器：<br><br><code>MSPLIM_NS</code> 表示非安全状态， <code>MSPLIM_S</code> 表示安全状态。                                                          |
| <code>PSPLIM</code>    | 栈限制寄存器分别限制 MSP 和 PSP 寄存器的最低有效地址。Cortex-M33 处理器中存在两个 PSPLIM 寄存器：<br><br><code>PSPLIM_NS</code> 表示非安全状态， <code>PSPLIM_S</code> 表示安全状态。                                                          |
| <code>LR (R14)</code>  | 链接寄存器（LR）即寄存器 <code>R14</code> 。其存储子程序、函数调用及异常的返回地址信息。                                                                                                                                        |
| <code>PC (R15)</code>  | 程序计数器（PC）为寄存器 <code>R15</code> 。其包含当前程序地址。<br>。                                                                                                                                               |

| 名称        | 描述                                                                                                                                                        |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| PSR       | 程序状态寄存器（PSR）综合了应用程序状态寄存器（APSР）、中断程序状态寄存器（IPSR）以及执行程序状态寄存器（EPSR）。这些寄存器提供了PSR的不同视角。                                                                         |
| PRIMASK   | PRIMASK寄存器防止优先级可配置的异常被激活。在包含Armv8-M安全扩展时，Cortex-M33处理器内设有两个PRIMASK寄存器：非安全状态使用PRIMASK_NS，安全状态使用PRIMASK_S。                                                  |
| BASEPRI   | BASEPRI寄存器定义异常处理的最低优先级门槛。Cortex-M33处理器内设有两个BASEPRI寄存器：非安全状态使用BASEPRI_NS，安全状态使用BASEPRI_S。                                                                  |
| FAULTMASK | FAULTMASK寄存器可防止激活除非屏蔽中断（Non-Maskable Interrupt, NMI）和安全硬故障（Secure HardFault）之外的所有异常。Cortex-M33处理器中设有两个FAULTMASK寄存器：FAULTMASK_NS用于非安全状态，FAULTMASK_S用于安全状态。 |
| CONTROL   | CONTROL寄存器控制处理器在线程模式下所使用的堆栈，并可选择控制特权级别。Cortex-M33处理器中设有两个CONTROL寄存器：CONTROL_NS用于非安全状态，CONTROL_S用于安全状态。                                                    |

### 3.7.4.6. 异常

异常由处理器和嵌套向量中断控制器（NVIC）处理并排序优先级。除架构定义的行为外，处理器实现了先进的异常和中断处理机制，以降低中断延迟，并包含实现定义的行为。

处理器核心与嵌套向量中断控制器（NVIC）共同对所有异常进行优先级排序和处理。

处理异常时：

- 所有异常均在处理程序模式中进行处理。
- 在异常发生时，处理器状态会自动保存到堆栈中，并在中断服务例程（ISR）结束时自动从堆栈恢复。
- 向量在状态保存的同时被并行获取，从而实现高效的中断入口。

处理器支持尾部链式调用，使连续中断无需状态保存和恢复的开销。

软件可选择仅启用配置数量中的部分中断，也可选择使用配置优先级的位数。

异常可指定为安全（Secure）或非安全（Non-secure）。当异常发生时，处理器会切换到相应的安全状态。安全异常与非安全异常的优先级可独立进行编程设置。您可以通过 [AIRCR.PRIS](#)位域降低非安全可配置异常的优先级，以使安全中断优先获得处理。

进入和返回异常时，寄存器状态始终使用与后台安全状态关联的堆栈指针保存。当从安全状态进入非安全异常时，所有寄存器状态都会被压栈，然后寄存器被清零，以防止安全数据被非安全处理程序访问。向量基址

地址介于安全态与非安全态之间。`VTOR_S`包含安全向量基地址，`VTOR_NS`包含非安全向量基地址。这些寄存器可以由软件编程，也可由系统在复位时初始化。

### **① 注意**

向量表条目支持Arm与Thumb指令间的交互操作。此操作导致向量值的第0位在异常进入时加载至执行程序状态寄存器（EPSR）的T位。向量表中所有已填充的向量其第0位必须置位。若表项第0位清零，将在对应向量的处理程序首条指令处引发`INVSTATE`故障。

#### 3.7.4.7. 安全归属与内存保护

处理器中的安全归属与内存保护由安全归属单元（SAU）及内存保护单元（MPU）提供。

SAU是一种可编程单元，用于确定地址的安全属性。RP2350包含8个内存区域。

对于指令和数据，SAU返回与地址相关联的安全属性。

对于指令，该属性决定执行该指令时处理器允许的安全状态。

它还可以识别是否可以从非安全状态调用位于安全地址的代码。

对于数据，该属性决定内存地址是否可从非安全状态访问，以及外部存储访问请求是否标记为安全或非安全。

若非安全状态访问标记为安全的地址，则处理器将触发`SecureFault`异常。若安全状态访问标记为非安全的地址，则相关内存访问将被标记为非安全。

SAU返回的安全级别为内部SAU（如配置）定义的区域类型与相关实现定义属性单元（IDAU）返回类型的组合。若地址映射至由内部和外部归属单元共同定义的区域，则选择安全级别最高的区域。

寄存器字段`SAU_CTRL.EN`和`SAU_CTRL.ALLNS`控制SAU的启用状态及SAU禁用时的默认安全级别。当`SAU_CTRL.EN`和`SAU_CTRL.ALLNS`均复位为零时，SAU被禁用，且除PPB空间内某些特定区域外，所有内存均被设置为安全级别。若SAU未启用且`SAU_CTRL.ALLNS`为零，则IDAU无法将任何内存区域设置为低于安全级别的状态，例如`Secure NSC`或`NS`。若SAU启用，则`SAU_CTRL.ALLNS`不影响内存安全级别。

RP2350支持Armv8-M受保护内存系统架构（PMSA）。MPU全面支持以下功能：

- 保护区域
- 访问权限
- 向系统导出内存属性

MPU不匹配和权限违规将触发`MemManage`中断处理程序。更多信息，请参阅Armv8-M架构参考手册。

您可以使用MPU来：

- 强制执行权限规则
- 划分进程
- 管理内存属性

MPU支持16个内存区域：8个安全区和8个非安全区。MPU在安全与非安全状态之间进行分区。安全和非安全MPU中的区域数量可独立配置，且每个区域均可编程以保护对应安全状态的内存。

### 3.7.4.8. 外部协处理器

外部协处理器接口：

- 支持从处理器至加速器组件及反向的低延迟数据传输。
- 其持续带宽最高可达处理器内存接口的两倍。

支持以下指令类型：

- 从Cortex-M33处理器向协处理器传输寄存器 **MCR**, **MCRR**, **MCR2**, **MCRR2**。
- 从协处理器到Cortex-M33处理器的寄存器转移 **MRC**、 **MRRC**、 **MRC2**、 **MRRC2**。
- 数据处理指令 **CDP**、 **CDP2**。

#### **i 注意**

协处理器指令的常规形式和扩展形式，例如 **MCR** 和 **MCRR2**，功能相同，但编码不同。**MRC** 和 **MRC2** 指令在处理器寄存器字段设置为PC时支持传输APS.R.NZV.C标志，例如 Rt == 0xF。

#### 3.7.4.8.1. 限制

协处理器指令适用以下限制：

- 不支持 **LDC(2)** 或 **STC(2)** 指令。若软件中包含 **<coproc>** 字段设置为0至7之间的值，且在 **CPACR/ NSACR** 寄存器相应字段启用协处理器，Cortex-M33处理器将始终尝试触发“未定义指令”(**UNDEFINSTR**)使用错误(**UsageFault**)异常。
- 数据传输指令的处理器寄存器字段不得包括堆栈指针(**Rt == 0xD**)，该编码在Armv8-M架构中为**不可预测**，并导致**CPACR/NSACR**寄存器中产生未定义指令(**UNDEFINSTR**) **UsageFault**异常。
- 如果在对应协处理器于 **CPACR/NSACR** 寄存器中被禁用时执行任何协处理器指令，Cortex-M33处理器将始终尝试产生一例无协处理器(**NOCP**) **UsageFault**异常。

#### 3.7.4.8.2. 数据传输速率

下表显示了协处理器接口的理想数据传输速率。这表明协处理器能够立即响应指令。若连续执行对应协处理器指令，则理想数据传输速率可持续保持。

以下指令的数据显示数据传输速率如下：

##### **MCR, MCR2 (处理器到协处理器)**

每周期32位

##### **MRC, MRC2 (协处理器至处理器)**

每周期32位

##### **MCRR, MCRR2 (处理器至协处理器)**

每周期64位

##### **MRRC, MRRC2 (协处理器至处理器)**

每周期64位

### 3.7.4.9. 执行时序

本节说明各类Cortex-M33指令的执行时间。该结果基于测量

有限且非特殊的一组常见指令示例，因此可能无法覆盖某些较为特殊的情况。

测量条件如下：

- 仅有一个内核运行
- 无缓存未命中（尤其是无XIP缓存未命中）
- 无活动的DMA操作

上述任一条件均可能影响指令取指及加载和存储操作的时序。有关存储器访问总线竞争的情况，请参见本数据手册其他章节中总线结构的描述。

#### 3.7.4.9.1. 结果延时

某些指令会生成两周期延迟的结果。使用此类结果作为后续指令的源操作数，将导致一周期的结果使用惩罚。任何指令的大多数输入参数均视为源操作数，包括：

- 数据处理（ALU）指令中的任何源寄存器
- 由 **LDR** 或 **LDM** 用于地址生成的任何寄存器（包括 **POP** 指令中的 **R13**）
- 由 **STR** 或 **STM** 用于地址生成的任何寄存器（但不包括待存储寄存器）（包括 **PUSH** 指令中的 **R13**）。

下例展示了一个加载指令后紧跟一条数据处理指令的指令序列，该序列会导致此惩罚：

```
LDR R0, [R1]
ADD R1, R0, R2
```

以下指令生成两周期延迟的结果：

- 某些数据处理指令中非简单移位所产生的目标寄存器（详见下文说明）
- 乘法指令的一个或多个目标寄存器
- **LDR** 指令的目标寄存器
- 除非该寄存器为 **R15**，否则 **LDM** 或 **POP** 指令寄存器列表中的最后一个寄存器

将上述指令的结果用作另一指令的源操作数时，操作间会产生一个周期的惩罚延迟。

#### 3.7.4.9.2. 简单算术与逻辑指令

大多数数据处理指令均在单个周期内完成。某些复杂操作（包括上述列举的及 **SEL** 指令）会产生结果使用延迟惩罚。

复杂操作须满足以下至少一项条件：

- 移位操作数且移位方式非 **LSL#0**、**LSL#1**、**LSL#2** 或 **LSL#3**
- 包含移位的立即数操作数（即不符合 **0x000000XY**、**0xXYXYXYXY**、**0x00XY00XY** 或 **0xXY00XY00** 格式）

当复杂指令带有用于设置标志的 **-S** 后缀时，即使下一条指令不依赖该标志，也必然产生一个周期的惩罚延迟。

以下操作不产生惩罚：

```
AND R0, R1, R2, LSL#4
MOV R3, R4
```

然而，以下操作确实产生惩罚：

```
AND R0, R1, R2, LSL#4
MOV R3, R0
```

```
ANDS R0, R1, R2, LSL#4
MOV R3, R4
```

**ADD** 与 **SUB** 提供了带有12位纯立即数操作数的变体。这些不产生惩罚。

带立即数操作数的**MOV**与 **MVNS**（包括带16位纯立即数操作数的**MOV**）不产生结果使用惩罚。

尽管与带移位操作数的逻辑操作类似，**UBFX**、**SBFX**及**BFI**不产生结果使用惩罚。

简单移位指令（**LSL**、**LSR**、**ASR**及**ROR**，移位量可指定为立即数常量或寄存器）均为单周期执行，且不产生结果使用惩罚。

#### 3.7.4.9.3. 乘法指令

乘法和乘法-累加指令在单周期内执行，但其结果有一个周期的延迟。

然而，乘法指令的结果作为后续乘法-累加指令的累加输入这一特殊情况不会导致一个周期的延迟。因此，假定满足以下所有条件，重复的乘法-累加操作可实现每周期一次的执行：

- 操作累积于同一寄存器或寄存器对
- 乘数和被乘数操作数来源于其他寄存器

如以下序列可实现每周期执行一条指令：

```
MLA R0, R1, R2, R3
MLA R3, R1, R2, R0
MLA R0, R1, R2, R3
MLA R3, R1, R2, R0
...

```

```
UMLAL R0, R1, R4, R5
UMLAL R2, R3, R6, R7
UMLAL R0, R1, R4, R5
UMLAL R2, R3, R6, R7
...

```

乘法器要求其乘法操作数于周期  $n$  输入，累加操作数（如有）于周期  $n+1$  输入，结果在周期  $n+2$  可用。

作为进一步示例，以下序列可在4个周期内完成：

```

ADD R2, R0, R1, LSL#23
MLA R3, R4, R5, R2
MOV R6, R3

```

**ADD** 指令通常会导致一个周期的结果使用延迟惩罚，但在本例中，其结果直到乘累加操作的第二个周期才需要，因此消除了该惩罚。

#### 3.7.4.9.4. 除法指令

设  $n$  为被除数和除数绝对值中最高有效位1的位置差。如果  $n$  为负（此时结果为零），除法操作需耗费2个周期。

否则，除法操作需耗费  $4+n/4$  个周期，向下取整。

将除法结果用作下一条指令输入时会产生一个周期的结果使用延迟惩罚。

#### 3.7.4.9.5. 寄存器加载指令（LDR和LDM）

加载操作对每个寄存器执行一个周期，外加可能一周期的结果延迟。如果被寻址的内存无法立即响应读取请求，例如由于指令预取产生的冲突，加载速度可能会变慢。

从结果延迟的角度来看，任何用于地址生成的寄存器均视为源操作数。示例详见表116。

表116 加载  
指令源  
操作数示例

| 指令               | 源操作数   | 非源操作数          |
|------------------|--------|----------------|
| LDR R0,[R5,R6]   | R5, R6 | R0             |
| LDMIA R7,{R0-R3} | R7     | R0, R1, R2, R3 |

与 **LDR** 指令的目标寄存器相关联存在一个周期的结果延迟，且 **LDM** 或 **POP** 指令寄存器列表中的最后一个寄存器也存在一个周期的结果延迟。例如，**R7** 在 **LDR R7,[R5,R6]** 和 **LDMIA R0,{R1-R7}** 中均存在一个周期的结果延迟。

后者情况下，**R1** 至 **R6** 寄存器不产生结果延迟。

加载 **R15** 不会引起任何结果延迟；但如第3.7.4.9.7节所述，将需要额外的周期。

#### 3.7.4.9.6. 寄存器存储（STR 和 STM 指令）

存储，包括依赖三个不同寄存器内容的指令，如 **STR R0,[R1,R2,LSL#2]**，均在一个周期内执行完成。与加载类似，如果目标内存无法立即接受请求，存储操作将变慢。

从结果延迟角度看，参与地址生成但非存储目标的寄存器计为源操作数。示例详见表117。

表117。寄存器  
存储源操作数  
示例

| 指令                | 源操作数   | 非源操作数          |
|-------------------|--------|----------------|
| STR R5,[R6,R7]    | R6, R7 | R5             |
| STMFD R5!,{R0-R3} | R5     | R0, R1, R2, R3 |

#### 3.7.4.9.7. 分支

本节涵盖任何可能修改 **R15** 的指令，包括以下内容：

- **MOV R15,Rx**

- BNE
- BL
- BX
- LDR R15, …
- POP {…,R15}

当跳转由加载指令引起 (LDR R15, …, LDMxx Rx, {…,R15}, 或POP {…,R15}) 时, 指令的基本耗时即为加载指令本身所需时间, 详见第3.7.4.9.5节。

对于其他可修改 R15 的指令 (MOV R15,Rx, B<cond>, BL, BX) , 指令的基本耗时为零。

实际发生跳转所需的总时间为基本耗时 + 2 + L+U- (K&F) 个周期; 未发生跳转时所需时间为基本耗时 + 1 - F 个周期, 其中 L、U、F、K 的取值均为0或1, 具体如下:

#### L

当分支来源于加载指令时为1 (如LDR R15,[R6]、LDMIA R13,{R0-R3,R15}等) ; 否则为0。

#### U

当所有以下条件均满足时, 值为1:

- 分支的目标地址未对齐到字边界
- 该地址处的指令长度为32位
- 紧接分支前执行的指令既非 POP 也非 PUSH

若上述任一条件不满足, 则 U 为0。

#### F

指示该分支是否可与上一条指令 *PrevInst* (即紧接分支执行的指令) 实现双发射 (亦称“折叠”)。当所有以下条件均满足时, F 为1:

- 分支指令为 B、B<cond>或BX R14 (但不包括 BX 跳转到其他寄存器, 亦不包括MOV R15,R14)
- *PrevInst* 长度为16位
- *PrevInst* 按顺序执行 (即自身未发生分支跳转), 或 *PrevInst* 为字对齐
- *PrevInst* 自身未与先前指令合并

如上述任一条件不成立, F 即为0。

#### K

当已确认分支将在执行 *PrevInst* 之前执行时为1; 否则为0。换言之, K 为1, 除非该分支为条件分支且 *PrevInst* 设置了标志位。

例如, 以下延迟循环共耗时299个周期:

```
10000002: MOVS R5,#100
10000004: SUBS R5,R5,#1
10000006: BNE 0x10000004
```

这些周期基于如下时序:

- MOVS R5,#100 指令耗时1个周期
- 每条SUBS R5,R5,#1 指令耗时1个周期
- 前99条BNE 指令每条耗时2个周期 (L=U=0, F=1, K=0)
- 最后一次未执行的 BNE 指令耗时为0周期 (L=U=0, F=1, K=0, 使用公式1-F)

在不同对齐情况下, 相同的延时循环耗时300个周期:

```

10000000: MOVS R5, #100
10000002: SUBS R5, R5, #1
10000004: BNE 0x10000002

```

这些周期基于如下时序：

- 执行 `MOVS R5, #100` 指令耗时 1 周期；
- 每执行一次 `SUBS R5, R5, #1` 指令耗时 1 周期；
- 首次执行 `BNE` 指令耗时 2 周期 ( $L=U=0, F=1, K=0$ )；
- 之后的 98 次 `BNE` 指令每次耗时 2 周期 ( $L=U=0, F=K=0$ )；
- 最后一次未执行的 `BNE` 指令耗时 1 周期 ( $L=U=0, F=K=0$ ，使用公式  $1-F$ )。

此较长延迟循环同样消耗 300 个周期：

```

10000000: MOVS R5, #100
10000002: SUBS R5, R5, #1
10000004: MOV R1, R2
10000006: BNE 0x10000002

```

这些周期基于如下时序：

- 执行 `MOVS R5, #100` 指令耗时 1 周期；
- 每执行一次 `SUBS R5, R5, #1` 指令耗时 1 周期；
- 每条 `MOV R1, R2` 指令耗时 1 个周期；
- 前 99 条 `BNE` 指令各耗时 1 个周期 ( $L=U=0, F=K=1$ )；
- 最后一条未执行的 `BNE` 指令耗时 0 周期 ( $L=U=0, F=K=1$ ；采用公式  $1-F$ )。

本例说明，若能在循环结束测试与分支指令之间插入一条指令，执行时间有可能实现零净耗时。

另一种优化是确保分支目标地址为字对齐或为 16 位指令。从此角度看，`BL` 指令之后的任何指令均可视为分支目标，因为子程序返回指令会跳转至该处。

如果空间和缓存允许，展开循环并内联子程序可彻底避免分支开销。

未被执行的分支序列将在 0 周期和 1 周期之间交替。这相当于一系列 `NOP` 指令，这些指令同样可以被折叠。然而，这并不等同于 IT 块中条件失败的指令序列。

### 3.7.4.9.8. IT (if-then) 块

IT 块中条件失败的指令以单周期执行。

IT 块中条件成功的大多数指令，其周期数与其正常无条件状态下的周期数相同。

### 3.7.4.9.9. 双发射

当一条 16 位指令紧随 `NOP` 指令（操作码为 `0xBF00`，非 `0x46C0`）时，指令将被折叠，`NOP` 指令执行周期为零。在某些情况下，这有助于将分支目标地址对齐至字边界，而无需

执行时间惩罚。

当一条16位操作码紧随 **IT** 指令时， **IT** 指令以零周期执行。

当满足以下所有条件时，Cortex-M33核将 **NOP** 与前一条指令（**PrevInst**）合并：

- **PrevInst** 长度为16位
- **PrevInst** 为顺序执行（非跳转到的），或**PrevInst**为字对齐
- **PrevInst** 本身未与更早指令合并

未执行的分支在此意义上类似于 **NOP** 指令：可依据相同规则进行合并。有关采取与未采取的分支何时合并的详细内容，参见第3.7.4.9.7节。

当两条多周期指令合并时，相互之间最多重叠一个周期。

### 3.7.4.9.10. 浮点协处理器操作

本节描述涉及单精度浮点协处理器（FPU）的操作。有关GPIO协处理器、双精度协处理器及冗余协处理器的时序，详见第3.7.4.9.11节及本文件中相关协处理器的详细描述。

发出浮点指令会使整数核心占用一个周期。之后，整数核心可以继续执行其他非FPU操作而不受影响。

尝试发出另一条FPU指令时，将暂停执行，直至FPU准备好接收该指令。

下列表列出了各种FPU指令的执行时序：

- **VADD.F32**、**VSUB.F32**和**VMUL.F32**可在同一个周期内完成，但结果延迟一个周期。因此，以下示例序列以每条指令两个周期的速度执行：

```
VADD.F32 s0, s0, s2
VADD.F32 s0, s0, s3
VADD.F32 s0, s0, s4
VADD.F32 s0, s0, s5
...
...
```

然而，以下交错示例以每条指令一个周期的速度执行：

```
VADD.F32 s0, s0, s2
VADD.F32 s1, s1, s3
VADD.F32 s0, s0, s4
VADD.F32 s1, s1, s5
...
...
```

此外，只要无指令依赖其前驱指令结果，您可以任意交错 **VADD.F32**、**VSUB.F32**和**VMUL.F32**指令，以实现单周期执行。

- **VMLA.F32** 和 **VFMA.F32** 占用FPU 3个周期，外加一个周期的结果延迟。

然而，连续的**VMLA.F32**或**VFMA.F32**指令向同一寄存器累积时，可以实现每三周期执行一条指令。

- 当任务可以交叉执行时，分开的 **VMUL.F32**和 **VADD.F32**指令比单条 **VMLA.F32**指令更高效。
- **VDIV.F32** 和 **VSQRT.F32** 占用FPU 14个周期，外加一个周期的结果延迟。
- **VMOV.F32 Sx,Ry** （从整数寄存器向协处理器移动一个字）耗时一个周期。

- **VMOV.F32 Rx,Sy** （从协处理器向整数寄存器移动一个字）耗时一个周期，外加一个周期的结果延迟。
- **VMOV.F32 Sx,Sy** （协处理器寄存器间移动一个字）耗时一个周期。
- **VMOV.F64 Dx,Ry,Rz** （将两个字从整数寄存器移动至协处理器）占用浮点单元两周期，整数核心一周期。
- **VMOV.F64 Rx,Ry,Dz** （将两个字从协处理器移动至整数寄存器）占用浮点单元和整数核心两周期。

### 3.7.4.9.11. 其他协处理器操作

协处理器如果未准备好，可能会阻塞操作。欲了解更多信息，请参阅特定协处理器的文档。

下表详细说明了各类协处理器指令的执行时序：

- 假设无阻塞发生，**CDP**指令耗时一个周期。
- **MCR**指令（将一个字从整数寄存器移动至协处理器）耗时一个周期。
- **MRC**指令（将一个字从协处理器移动至整数寄存器）耗时一个周期，另加一个周期的结果延迟。
- **MCRR**指令（将两个字从整数寄存器移动至协处理器）耗时一个周期。
- 一条**MRRC**指令（将两个字从协处理器移动到整数寄存器）执行一个周期，外加一个周期的结果延迟。

### 3.7.4.9.12. 指令取指

每个Cortex-M33核心均具备独立的指令总线和数据总线（“哈佛架构”）。每个核心对内存的带宽为每周期32位。由于每条指令最长为32位，对于顺序执行的代码，指令预取器具备足够带宽，确保处理器核心始终拥有指令。

在RP2350中，当指令总线和数据总线尝试访问连接至AHB5交叉开关相同下游端口的内存时，可能发生争用。例如，从主SRAM执行的代码可能试图加载存储在附近的字面值。该加载可能与对同一SRAM的指令预取发生冲突。为降低此类冲突的可能性，主SRAM被划分为跨四字组的多个银行：地址模16不同的字被存储在不同的银行中。

由于预取器通常比执行提前约两个字（8字节）运行，这意味着一条指令如果读取自己前方8（模16）字节，可能会导致冲突。例如，以下指令读取前方40字节（因为此处**PC**表示下一条指令的地址），有时可能会导致一个周期的惩罚：

```
LDR R8, [PC, #32] @ 32位指令
```

### 3.7.4.10. 调试

Cortex-M33调试功能包括处理器暂停、单步执行、处理器核心寄存器访问、向量捕获、无限软件断点及完整系统内存访问。

处理器还支持在实现阶段配置的硬件断点和监视点：

- 支持八个指令比较器的断点单元
- 支持四个数据监视点比较器的监视单元

Cortex-M33处理器支持系统级调试认证，用以控制调试器对资源

及内存的访问。通过Armv8-M安全扩展进行的认证，可允许调试器完全访问非安全代码和数据，而不会泄露任何安全信息。

处理器实现可被划分为多个区域，以将调试组件置于独立于处理器核及NVIC的电源域中。

所有调试寄存器均可通过D-AHB接口访问。

更多信息，请参阅Armv8-M架构参考手册。

### 3.7.4.11. 数据观察点与跟踪单元（DWT）

DWT为全功能配置，包含四个比较器（[DWT\\_COMP0](#)至[DWT\\_COMP3](#)）。这些比较器支持以下功能：

- 硬件观察点支持
- 硬件跟踪数据包支持
- 支持ETM/MTB/CTI触发的CMPMATCH
- 支持周期计数匹配（仅限DWT\_COMP0）
- 支持指令地址匹配
- 支持数据地址匹配
- 支持数据值匹配（仅限简化DWT中的DWT\_COMP1，以及完整DWT中的DWT\_COMP3）
- 支持关联/限制匹配（仅限DWT\_COMP1和DWT\_COMP3）。DWT包

含以下计数器：

- 周期计数（[DWT\\_CYCCNT.CYCCNT](#)）
- 折叠指令计数（FOLDCNT）
- 执行所有加载/存储指令所需的额外周期（LSUCNT）
- 处理器休眠周期（SLEEP\_CNT）
- 执行多周期指令及指令取指阻塞所需的额外周期（CPICNT）
- 异常处理所用周期（EXCCNT）

使用DWT之前，请将[DEMCR.TRCENA](#)位设置为1。

DWT向ITM和TPIU周期性发送协议同步请求。

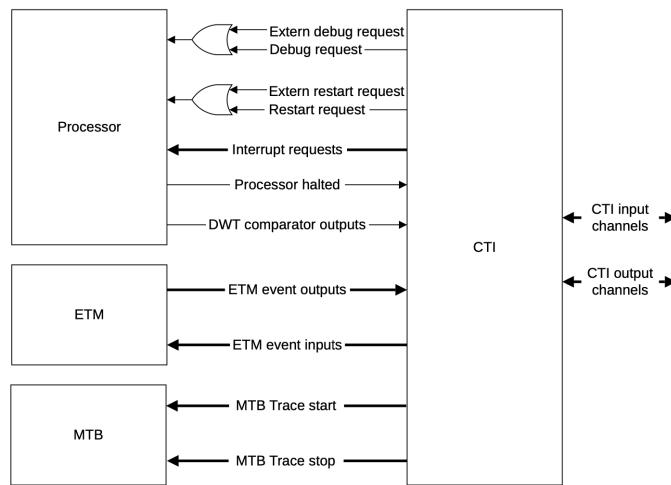
### 3.7.4.12. 交叉触发接口（CTI）

CTI使调试逻辑、MTB和ETM能够相互通信，并与其他CoreSight组件协作。这称为交叉触发。例如，您可以配置CTI，在ETM触发事件发生时生成中断，或在检测到DWT比较器匹配时启动跟踪。

下图展示了调试系统组件及可用的触发输入和输出：

[图15显示了调试系统的组成部分。](#)

图 15. 调试  
系统组件



下表显示了 CTI 触发输入如何连接至 Cortex-M33 处理器：

表 118. 触发  
信号至 CTI

| 信号           | 描述                   | 连接           | 确认，握手 |
|--------------|----------------------|--------------|-------|
| CTITRIGIN[7] |                      | ETM 至 CTI    | 脉冲型   |
| CTITRIGIN[6] |                      | ETM 至 CTI    | 脉冲型   |
| CTITRIGIN[5] | ETM 事件输出 1           | ETM 至 CTI    | 脉冲型   |
| CTITRIGIN[4] | ETM 事件输出 0 或 比较器输出 3 | ETM/处理器至 CTI | 脉冲型   |
| CTITRIGIN[3] | DWT 比较器输出 2          | 处理器至 CTI     | 脉冲型   |
| CTITRIGIN[2] | DWT 比较器输出 1          | 处理器至 CTI     | 脉冲型   |
| CTITRIGIN[1] | DWT 比较器输出 0          | 处理器至 CTI     | 脉冲型   |
| CTITRIGIN[0] | 处理器已停止               | 处理器至 CTI     | 脉冲型   |

下表显示了 CTI 触发输出与处理器及 ETM 的连接方式：

表 119 来自 CTI  
的触发信号

| 信号            | 描述                    | 连接              | 确认，握手                                         |
|---------------|-----------------------|-----------------|-----------------------------------------------|
| CTITRIGOUT[7] | ETM 事件输入 3            | CTI 至 ETM       | 脉冲型                                           |
| CTITRIGOUT[6] | ETM 事件输入 2            | CTI 至 ETM       | 脉冲型                                           |
| CTITRIGOUT[5] | ETM 事件输入 1 或 MTB 跟踪停止 | CTI 至 ETM 或 MTB | 脉冲型                                           |
| CTITRIGOUT[4] | ETM 事件输入 1 或 MTB 跟踪启动 | CTI 至 ETM 或 MTB | 脉冲型                                           |
| CTITRIGOUT[3] | 中断请求 1                | CTI 至系统         | 通过在 ISR 中写入 <a href="#">CTIINTACK</a> 寄存器予以确认 |
| CTITRIGOUT[2] | 中断请求 0                | CTI 至系统         | 通过在 ISR 中写入 <a href="#">CTIINTACK</a> 寄存器予以确认 |
| CTITRIGOUT[1] | 处理器重启                 | CTI 至处理器        | 处理器已重启                                        |

| 信号            | 描述      | 连接      | 确认，握手                                     |
|---------------|---------|---------|-------------------------------------------|
| CTITRIGOUT[0] | 处理器调试请求 | CTI至处理器 | 通过调试器写入 <a href="#">CTIINTACK</a> 寄存器予以确认 |

在使用CTI触发输出0使处理器停止后，处理器调试请求信号仍保持有效。调试器必须先写入[CTIINTACK](#)以清除停止请求，随后方可重新启动处理器。

在通过CTI触发输出1或2发起中断后，中断服务例程（ISR）必须通过写入CTI中断确认寄存器[CTIINTACK](#)以清除中断请求。

仅当处理器启用了侵入式调试时，来自CTI到系统的中断请求才会被断言。

### 3.7.4.12.1. CTI编程模型

下表列出了CTI可编程寄存器，包括每个寄存器的地址偏移、类型及复位值。

有关寄存器说明，请参阅Arm CoreSight™ SoC-400技术参考手册。

表120. Cortex-M33  
CTI寄存器概要

| 地址偏移                  | 名称               | 类型 | 复位值        | 描述             |
|-----------------------|------------------|----|------------|----------------|
| 0xE0042000            | CTICONTROL       | 读写 | 0x00000000 | CTI控制寄存器       |
| 0xE0042010            | CTIINTACK        | WO | 未知         | CTI中断确认寄存器     |
| 0xE0042014            | CTIAPPSET        | 读写 | 0x00000000 | CTI应用触发设置寄存器   |
| 0xE0042018            | CTIAPPCLEAR      | 读写 | 0x00000000 | CTI 应用触发清除寄存器  |
| 0xE004201C            | CTIAPPPULSE      | WO | 未知         | CTI 应用脉冲寄存器    |
| 0xE0042020-0xE004203C | CTIINEN[7:0]     | 读写 | 0x00000000 | CTI 触发至通道使能寄存器 |
| 0xE00420A0-0xE00420BC | CTIOUTEN[7:0]    | 读写 | 0x00000000 | CTI 通道至触发使能寄存器 |
| 0xE0042130            | CTITRIGINSTATUS  | 只读 | 0x00000000 | CTI 输入触发状态寄存器  |
| 0xE0042134            | CTITRIGOUTSTATUS | 只读 | 0x00000000 | CTI 输出触发状态寄存器  |
| 0xE0042138            | CTICHINSTATUS    | 只读 | 0x00000000 | CTI 通道输入状态寄存器  |
| 0xE0042140            | CTIGATE          | 读写 | 0x0000000F | 启用 CTI 通道门控寄存器 |
| 0xE0042144            | ASICCTL          | 读写 | 0x00000000 | 外部多路复用器控制寄存器   |
| 0xE0042EE4            | ITCHOUT          | WO | 未知         | 集成测试通道输出寄存器    |
| 0xE0042EE8            | ITTRIGOUT        | WO | 未知         | 集成测试触发输出寄存器    |
| 0xE0042EF4            | ITCHIN           | 只读 | 0x00000000 | 集成测试通道输入寄存器    |
| 0xE0042F00            | ITCTRL           | 读写 | 0x00000000 | 集成模式控制寄存器      |
| 0xE0042FC8            | DEVID            | 只读 | 0x00040800 | 设备配置寄存器        |
| 0xE0042FBC            | DEVARCH          | 只读 | 0x47701A14 | 设备架构寄存器        |

| 地址偏移       | 名称      | 类型 | 复位值        | 描述        |
|------------|---------|----|------------|-----------|
| 0xE0042FCC | DEVTYPE | 只读 | 0x00000014 | 设备类型标识寄存器 |
| 0xE0042FD0 | PIDR4   | 只读 | 0x00000004 | 外设ID4寄存器  |
| 0xE0042FD4 | PIDR5   | 只读 | 0x00000000 | 外设ID5寄存器  |
| 0xE0042FD8 | PIDR6   | 只读 | 0x00000000 | 外设ID6寄存器  |
| 0xE0042FDC | PIDR7   | 只读 | 0x00000000 | 外设ID7寄存器  |
| 0xE0042FE0 | PIDR0   | 只读 | 0x00000021 | 外设ID0寄存器  |
| 0xE0042FE4 | PIDR1   | 只读 | 0x000000BD | 外设ID1寄存器  |
| 0xE0042FE8 | PIDR2   | 只读 | 0x0000000B | 外设ID2寄存器  |
| 0xE0042FEC | PIDR3   | 只读 | 0x00000001 | 外设ID3寄存器  |
| 0xE0042FF0 | CIDR0   | 只读 | 0x0000000D | 组件ID0寄存器  |
| 0xE0042FF4 | CIDR1   | 只读 | 0x00000090 | 组件ID1寄存器  |
| 0xE0042FF8 | CIDR2   | 只读 | 0x00000005 | 组件ID2寄存器  |
| 0xE0042FFC | CIDR3   | 只读 | 0x000000B1 | 组件ID3寄存器  |

### 3.7.5. 寄存器列表

Arm Cortex-M33 寄存器起始于基地址 `0xe0000000`, 在 SDK 中定义为 `PPB_BASE`。

表121 M33寄存器列表

| 偏移量     | 名称         | 说明             |
|---------|------------|----------------|
| 0x00000 | ITM_STIM0  | ITM 刺激端口寄存器 0  |
| 0x00004 | ITM_STIM1  | ITM 刺激端口寄存器 1  |
| 0x00008 | ITM_STIM2  | ITM 刺激端口寄存器 2  |
| 0x0000c | ITM_STIM3  | ITM 刺激端口寄存器 3  |
| 0x00010 | ITM_STIM4  | ITM 刺激端口寄存器 4  |
| 0x00014 | ITM_STIM5  | ITM 刺激端口寄存器 5  |
| 0x00018 | ITM_STIM6  | ITM 刺激端口寄存器 6  |
| 0x0001c | ITM_STIM7  | ITM 刺激端口寄存器 7  |
| 0x00020 | ITM_STIM8  | ITM 刺激端口寄存器 8  |
| 0x00024 | ITM_STIM9  | ITM 刺激端口寄存器 9  |
| 0x00028 | ITM_STIM10 | ITM 刺激端口寄存器 10 |
| 0x0002c | ITM_STIM11 | ITM 刺激端口寄存器 11 |
| 0x00030 | ITM_STIM12 | ITM 刺激端口寄存器 12 |
| 0x00034 | ITM_STIM13 | ITM 刺激端口寄存器 13 |
| 0x00038 | ITM_STIM14 | ITM 刺激端口寄存器 14 |
| 0x0003c | ITM_STIM15 | ITM 刺激端口寄存器 15 |
| 0x00040 | ITM_STIM16 | ITM 刺激端口寄存器 16 |
| 0x00044 | ITM_STIM17 | ITM 刺激端口寄存器 17 |

| 偏移量     | 名称          | 说明                       |
|---------|-------------|--------------------------|
| 0x00048 | ITM_STIM18  | ITM 刺激端口寄存器 18           |
| 0x0004c | ITM_STIM19  | ITM 刺激端口寄存器 19           |
| 0x00050 | ITM_STIM20  | ITM 刺激端口寄存器 20           |
| 0x00054 | ITM_STIM21  | ITM 刺激端口寄存器 21           |
| 0x00058 | ITM_STIM22  | ITM 刺激端口寄存器 22           |
| 0x0005c | ITM_STIM23  | ITM 刺激端口寄存器 23           |
| 0x00060 | ITM_STIM24  | ITM 刺激端口寄存器 24           |
| 0x00064 | ITM_STIM25  | ITM 刺激端口寄存器 25           |
| 0x00068 | ITM_STIM26  | ITM 刺激端口寄存器 26           |
| 0x0006c | ITM_STIM27  | ITM 刺激端口寄存器 27           |
| 0x00070 | ITM_STIM28  | ITM 刺激端口寄存器 28           |
| 0x00074 | ITM_STIM29  | ITM 刺激端口寄存器 29           |
| 0x00078 | ITM_STIM30  | ITM 刺激端口寄存器 30           |
| 0x0007c | ITM_STIM31  | ITM 刺激端口寄存器 31           |
| 0x00e00 | ITM_TER0    | 为每个 ITM_STIM 寄存器配置独立的使能位 |
| 0x00e40 | ITM_TPR     | 控制非特权代码可访问的刺激端口          |
| 0x00e80 | ITM_TCR     | 配置并控制通过ITM接口的传输          |
| 0x00ef0 | INT_ATREADY | 集成模式：读取ATB就绪信号           |
| 0x00ef8 | INT_ATVALID | 集成模式：写入ATB有效信号           |
| 0x00f00 | ITM_ITCTRL  | 集成模式控制寄存器                |
| 0x00fb0 | ITM_DEVARCH | 提供ITM的CoreSight发现信息      |
| 0x00fcc | ITM_DEVTYPE | 提供ITM的CoreSight发现信息      |
| 0x00fd0 | ITM_PIDR4   | 提供ITM的CoreSight发现信息      |
| 0x00fd4 | ITM_PIDR5   | 提供ITM的CoreSight发现信息      |
| 0x00fd8 | ITM_PIDR6   | 提供ITM的CoreSight发现信息      |
| 0x00fdc | ITM_PIDR7   | 提供ITM的CoreSight发现信息      |
| 0x00fe0 | ITM_PIDR0   | 提供ITM的CoreSight发现信息      |
| 0x00fe4 | ITM_PIDR1   | 提供ITM的CoreSight发现信息      |
| 0x00fe8 | ITM_PIDR2   | 提供ITM的CoreSight发现信息      |
| 0x00fec | ITM_PIDR3   | 提供ITM的CoreSight发现信息      |
| 0x00ff0 | ITM_CIDR0   | 提供ITM的CoreSight发现信息      |
| 0x00ff4 | ITM_CIDR1   | 提供ITM的CoreSight发现信息      |
| 0x00ff8 | ITM_CIDR2   | 提供ITM的CoreSight发现信息      |
| 0x00ffc | ITM_CIDR3   | 提供ITM的CoreSight发现信息      |

| 偏移量     | 名称            | 说明                                         |
|---------|---------------|--------------------------------------------|
| 0x01000 | DWT_CTRL      | 提供DWT单元的配置和状态信息，并用于控制该单元功能                 |
| 0x01004 | DWT_CYCCNT    | 显示或设置处理器周期计数器CYCCNT的数值                     |
| 0x0100c | DWT_EXCCNT    | 统计异常处理期间消耗的总周期数                            |
| 0x01014 | DWT_LSUCNT    | 递增计数以反映执行所有加载或存储指令所需的额外周期                  |
| 0x01018 | DWT_FOLDCNT   | 递增计数以反映执行所有加载或存储指令所需的额外周期                  |
| 0x01020 | DWT_COMPO     | 为观察点比较器0提供参考值                              |
| 0x01028 | DWT_FUNCTION0 | 控制观察点比较器0的运行                               |
| 0x01030 | DWT_COMP1     | 为监视点比较器1提供参考值                              |
| 0x01038 | DWT_FUNCTION1 | 控制监视点比较器1的操作                               |
| 0x01040 | DWT_COMP2     | 为监视点比较器2提供参考值                              |
| 0x01048 | DWT_FUNCTION2 | 控制监视点比较器2的操作                               |
| 0x01050 | DWT_COMP3     | 为监视点比较器3提供参考值                              |
| 0x01058 | DWT_FUNCTION3 | 控制监视点比较器3的操作                               |
| 0x01fbc | DWT_DEVARCH   | 提供DWT的CoreSight发现信息                        |
| 0x01fcc | DWT_DEVTYPE   | 提供DWT的CoreSight发现信息                        |
| 0x01fd0 | DWT_PIDR4     | 提供DWT的CoreSight发现信息                        |
| 0x01fd4 | DWT_PIDR5     | 提供DWT的CoreSight发现信息                        |
| 0x01fd8 | DWT_PIDR6     | 提供DWT的CoreSight发现信息                        |
| 0x01fdc | DWT_PIDR7     | 提供DWT的CoreSight发现信息                        |
| 0x01fe0 | DWT_PIDR0     | 提供DWT的CoreSight发现信息                        |
| 0x01fe4 | DWT_PIDR1     | 提供DWT的CoreSight发现信息                        |
| 0x01fe8 | DWT_PIDR2     | 提供DWT的CoreSight发现信息                        |
| 0x01fec | DWT_PIDR3     | 提供DWT的CoreSight发现信息                        |
| 0x01ff0 | DWT_CIDR0     | 提供DWT的CoreSight发现信息                        |
| 0x01ff4 | DWT_CIDR1     | 提供DWT的CoreSight发现信息                        |
| 0x01ff8 | DWT_CIDR2     | 提供DWT的CoreSight发现信息                        |
| 0x01ffc | DWT_CIDR3     | 提供DWT的CoreSight发现信息                        |
| 0x02000 | FP_CTRL       | 提供FPB实现信息及FPB单元的全局启用                       |
| 0x02004 | FP_REMAP      | 指示实现是否支持Flash Patch重映射，若支持，则保存重映射目标地址      |
| 0x02008 | FP_COMPO      | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |

| 偏移量     | 名称         | 说明                                         |
|---------|------------|--------------------------------------------|
| 0x0200c | FP_COMP1   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x02010 | FP_COMP2   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x02014 | FP_COMP3   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x02018 | FP_COMP4   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x0201c | FP_COMP5   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x02020 | FP_COMP6   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x02024 | FP_COMP7   | 保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器 |
| 0x02fbc | FP_DEVARCH | 提供FPB的CoreSight发现信息                        |
| 0x02fcc | FP_DEVTYPE | 提供FPB的CoreSight发现信息                        |
| 0x02fd0 | FP_PIDR4   | 提供FP的CoreSight发现信息                         |
| 0x02fd4 | FP_PIDR5   | 提供FP的CoreSight发现信息                         |
| 0x02fd8 | FP_PIDR6   | 提供FP的CoreSight发现信息                         |
| 0x02fdc | FP_PIDR7   | 提供FP的CoreSight发现信息                         |
| 0x02fe0 | FP_PIDR0   | 提供FP的CoreSight发现信息                         |
| 0x02fe4 | FP_PIDR1   | 提供FP的CoreSight发现信息                         |
| 0x02fe8 | FP_PIDR2   | 提供FP的CoreSight发现信息                         |
| 0x02fec | FP_PIDR3   | 提供FP的CoreSight发现信息                         |
| 0x02ff0 | FP_CIDR0   | 提供FP的CoreSight发现信息                         |
| 0x02ff4 | FP_CIDR1   | 提供FP的CoreSight发现信息                         |
| 0x02ff8 | FP_CIDR2   | 提供FP的CoreSight发现信息                         |
| 0x02ffc | FP_CIDR3   | 提供FP的CoreSight发现信息                         |
| 0x0e004 | ICTR       | 提供中断控制器相关信息                                |

| 偏移量     | 名称         | 说明                            |
|---------|------------|-------------------------------|
| 0x0e008 | ACTLR      | 提供实现定义的配置与控制选项                |
| 0x0e010 | SYST_CSR   | SysTick 控制与状态寄存器              |
| 0x0e014 | SYST_RVR   | SysTick 重载值寄存器                |
| 0x0e018 | SYST_CVR   | SysTick 当前值寄存器                |
| 0x0e01c | SYST_CALIB | SysTick 校准值寄存器                |
| 0x0e100 | NVIC_ISER0 | 启用或读取每组32个中断的启用状态             |
| 0x0e104 | NVIC_ISER1 | 启用或读取每组32个中断的启用状态             |
| 0x0e180 | NVIC_ICERO | 清除或读取每组32个中断的启用状态             |
| 0x0e184 | NVIC_ICER1 | 清除或读取每组32个中断的启用状态             |
| 0x0e200 | NVIC_ISPR0 | 启用或读取每组32个中断的挂起状态             |
| 0x0e204 | NVIC_ISPR1 | 启用或读取每组32个中断的挂起状态             |
| 0x0e280 | NVIC_ICP0  | 清除或读取每组32个中断的挂起状态             |
| 0x0e284 | NVIC_ICP1  | 清除或读取每组32个中断的挂起状态             |
| 0x0e300 | NVIC_IABR0 | 针对每组32个中断，显示每个中断的活动状态         |
| 0x0e304 | NVIC_IABR1 | 针对每组32个中断，显示每个中断的活动状态         |
| 0x0e380 | NVIC_ITNS0 | 针对每组32个中断，确定每个中断的目标是非安全态还是安全态 |
| 0x0e384 | NVIC_ITNS1 | 针对每组32个中断，确定每个中断的目标是非安全态还是安全态 |
| 0x0e400 | NVIC_IPR0  | 设置或读取中断优先级                    |
| 0x0e404 | NVIC_IPR1  | 设置或读取中断优先级                    |
| 0x0e408 | NVIC_IPR2  | 设置或读取中断优先级                    |
| 0x0e40c | NVIC_IPR3  | 设置或读取中断优先级                    |
| 0x0e410 | NVIC_IPR4  | 设置或读取中断优先级                    |
| 0x0e414 | NVIC_IPR5  | 设置或读取中断优先级                    |
| 0x0e418 | NVIC_IPR6  | 设置或读取中断优先级                    |
| 0x0e41c | NVIC_IPR7  | 设置或读取中断优先级                    |
| 0x0e420 | NVIC_IPR8  | 设置或读取中断优先级                    |
| 0x0e424 | NVIC_IPR9  | 设置或读取中断优先级                    |
| 0x0e428 | NVIC_IPR10 | 设置或读取中断优先级                    |
| 0x0e42c | NVIC_IPR11 | 设置或读取中断优先级                    |
| 0x0e430 | NVIC_IPR12 | 设置或读取中断优先级                    |
| 0x0e434 | NVIC_IPR13 | 设置或读取中断优先级                    |
| 0x0e438 | NVIC_IPR14 | 设置或读取中断优先级                    |

| 偏移量     | 名称         | 说明                                                                                                                                |
|---------|------------|-----------------------------------------------------------------------------------------------------------------------------------|
| 0x0e43c | NVIC_IPR15 | 设置或读取中断优先级                                                                                                                        |
| 0x0ed00 | CPUID      | 提供PE的识别信息，包括设备的实现者代码及设备ID编号                                                                                                       |
| 0x0ed04 | ICSR       | 控制并提供 NMI、PendSV、SysTick 及中断的状态信息                                                                                                 |
| 0x0ed08 | VTOR       | 向量表偏移寄存器                                                                                                                          |
| 0x0ed0c | AIRCR      | 应用中断与复位控制寄存器                                                                                                                      |
| 0x0ed10 | SCR        | 系统控制寄存器                                                                                                                           |
| 0x0ed14 | CCR        | 设置或返回配置与控制数据                                                                                                                      |
| 0x0ed18 | SHPR1      | 设置或返回系统处理程序 4 至 7 的优先级                                                                                                            |
| 0x0ed1c | SHPR2      | 设置或返回系统处理程序 8 至 11 的优先级                                                                                                           |
| 0x0ed20 | SHPR3      | 设置或返回系统处理程序 12 至 15 的优先级                                                                                                          |
| 0x0ed24 | SHCSR      | 提供系统异常的活动和挂起状态的访问                                                                                                                 |
| 0x0ed28 | CFSR       | 包含三个可配置故障状态寄存器<br><br>31:16 UFSR：提供有关 UsageFault 异常的信息<br><br>15:8 BFRS：提供有关 BusFault 异常的信息<br><br>7:0 MMFSR：提供有关 MemManage 异常的信息 |
| 0x0ed2c | HFSR       | 显示任何 HardFault 的原因                                                                                                                |
| 0x0ed30 | DFSR       | 显示发生的调试事件类型                                                                                                                       |
| 0x0ed34 | MMFAR      | 显示导致MPU故障的内存地址                                                                                                                    |
| 0x0ed38 | BFAR       | 显示与精确数据访问相关的地址<br>BusFault                                                                                                        |
| 0x0ed40 | ID_PFR0    | 提供关于PE所支持指令集的顶层信息<br>由PE提供                                                                                                        |
| 0x0ed44 | ID_PFR1    | 提供有关程序员模型及扩展支持的信息                                                                                                                 |
| 0x0ed48 | ID_DFR0    | 提供关于调试系统的顶层信息                                                                                                                     |
| 0x0ed4c | ID_AFR0    | 提供有关PE实现定义特性的说明                                                                                                                   |
| 0x0ed50 | ID_MMFR0   | 提供有关已实现的内存模型及内存管理支持的信息                                                                                                            |
| 0x0ed54 | ID_MMFR1   | 提供有关已实现的内存模型及内存管理支持的信息                                                                                                            |
| 0x0ed58 | ID_MMFR2   | 提供有关已实现的内存模型及内存管理支持的信息                                                                                                            |
| 0x0ed5c | ID_MMFR3   | 提供有关已实现的内存模型及内存管理支持的信息                                                                                                            |

| 偏移量     | 名称          | 说明                                                                                           |
|---------|-------------|----------------------------------------------------------------------------------------------|
| 0x0ed60 | ID_ISAR0    | 提供有关PE实现的指令集的信息                                                                              |
| 0x0ed64 | ID_ISAR1    | 提供有关PE实现的指令集的信息                                                                              |
| 0x0ed68 | ID_ISAR2    | 提供有关PE实现的指令集的信息                                                                              |
| 0x0ed6c | ID_ISAR3    | 提供有关PE实现的指令集的信息                                                                              |
| 0x0ed70 | ID_ISAR4    | 提供有关PE实现的指令集的信息                                                                              |
| 0x0ed74 | ID_ISAR5    | 提供有关PE实现的指令集的信息                                                                              |
| 0x0ed7c | CTR         | 提供有关缓存架构的信息当CLIDR为零时，CTR为保留值0（RES0）                                                          |
| 0x0ed88 | CPACR       | 指定协处理器及浮点扩展的访问权限                                                                             |
| 0x0ed8c | NSACR       | 定义浮点扩展及协处理器 CP0 至 CP7 的非安全访问权限                                                               |
| 0x0ed90 | MPU_TYPE    | MPU 类型寄存器指示 MPU `FTSSS 支持的区域数量                                                               |
| 0x0ed94 | MPU_CTRL    | 使能 MPU；当 MPU 使能时，控制是否将默认内存映射作为特权访问的背景区域，以及在 FAULTMASK 置为 1 时，是否对 HardFault、NMI 及异常处理程序启用 MPU |
| 0x0ed98 | MPU_RNR     | 选择当前由 MPU_RBAR 和 MPU_RLAR 访问的区域                                                              |
| 0x0ed9c | MPU_RBAR    | 提供对当前选定 MPU 区域 `FTSSS 基地址的间接读写访问                                                             |
| 0x0eda0 | MPU_RLAR    | 提供对当前选定 MPU 区域 `FTSSS 限制地址的间接读写访问                                                            |
| 0x0eda4 | MPU_RBAR_A1 | 提供对由 MPU_RNR[7:2]:(1[1:0]) 选择的 MPU 区域基地址的间接读写访问 `FTSSS                                       |
| 0x0eda8 | MPU_RLAR_A1 | 提供对由 MPU_RNR[7:2]:(1[1:0]) 选择的当前 MPU 区域限制地址的间接读写访问 `FTSSS                                    |
| 0x0edac | MPU_RBAR_A2 | 提供对由 MPU_RNR[7:2]:(2[1:0]) 选择的 MPU 区域基地址的间接读写访问 `FTSSS                                       |
| 0x0edb0 | MPU_RLAR_A2 | 提供对由 MPU_RNR[7:2]:(2[1:0]) 选择的当前 MPU 区域限制地址的间接读写访问 `FTSSS                                    |
| 0x0edb4 | MPU_RBAR_A3 | 提供对由 MPU_RNR[7:2]:(3[1:0]) 选择的 MPU 区域基地址的间接读写访问 `FTSSS                                       |

| 偏移量     | 名称          | 说明                                                                                 |
|---------|-------------|------------------------------------------------------------------------------------|
| 0x0edb8 | MPU_RLAR_A3 | 提供对当前由 MPU_RNR[7:2]:(3[1:0]) `FTSSS` 选择的 MPU 区域限制地址的间接读写访问                         |
| 0x0edc0 | MPU_MAIR0   | 与 MPU_MAIR1 一起，提供对应于 AttrIndex 值的内存属性编码                                            |
| 0x0edc4 | MPU_MAIR1   | 与 MPU_MAIR0 一起，提供对应于 AttrIndex 值的内存属性编码                                            |
| 0x0edd0 | SAU_CTRL    | 允许启用安全属性单元                                                                         |
| 0x0edd4 | SAU_TYPE    | 指示安全属性单元已实现的区域数量                                                                   |
| 0x0edd8 | SAU_RNR     | 选择当前由 SAU_RBAR 和 SAU_RLAR 访问的区域                                                    |
| 0x0eddc | SAU_RBAR    | 提供对当前选定 SAU 区域基址的间接读写访问                                                            |
| 0x0ede0 | SAU_RLAR    | 提供对当前选定 SAU 区域限制地址的间接读写访问                                                          |
| 0x0ede4 | SFSR        | 提供有关任一安全相关故障的信息                                                                    |
| 0x0ede8 | SFAR        | 显示引发安全违规的内存地址                                                                      |
| 0x0edf0 | DHCSR       | 控制调试中断                                                                             |
| 0x0edf4 | DCRSR       | 与 DCRSR 协同，提供对通用寄存器、特定用途寄存器及浮点扩展寄存器的调试访问。写入 DCRSR 以指定传输的寄存器、传输类型（读或写），并启动传输        |
| 0x0edf8 | DCRDR       | 与 DCRSR 协同，提供对通用寄存器、特定用途寄存器及浮点扩展寄存器的调试访问。若实现主扩展，还可用于外部调试器与运行于处理单元（PE）上的调试代理之间的消息传递 |
| 0x0edfc | DEMCR       | 管理调试过程中向量捕获行为及调试监视器的处理                                                             |
| 0x0ee08 | DSCSR       | 提供安全调试的控制与状态信息                                                                     |
| 0x0ef00 | STIR        | 为软件生成中断提供机制                                                                        |
| 0x0ef34 | FPCCR       | 存储浮点扩展的控制数据                                                                        |
| 0x0ef38 | FPCAR       | 存储异常堆栈帧上未使用的浮点寄存器空间位置                                                              |
| 0x0ef3c | FPDSCR      | 存储处理器在创建新的浮点上下文时赋予FPSCR的浮点状态控制数据默认值                                                |
| 0x0ef40 | MVFR0       | 描述浮点扩展所提供的特性                                                                       |
| 0x0ef44 | MVFR1       | 描述浮点扩展所提供的特性                                                                       |
| 0x0ef48 | MVFR2       | 描述浮点扩展所提供的特性                                                                       |
| 0x0efbc | DDEVARCH    | 为SCS提供CoreSight发现信息                                                                |

| 偏移量     | 名称            | 说明                                                        |
|---------|---------------|-----------------------------------------------------------|
| 0x0efcc | DDEVTYPE      | 为SCS提供CoreSight发现信息                                       |
| 0x0efd0 | DPIDR4        | 为SCS提供CoreSight发现信息                                       |
| 0x0efd4 | DPIDR5        | 为SCS提供CoreSight发现信息                                       |
| 0x0efd8 | DPIDR6        | 为SCS提供CoreSight发现信息                                       |
| 0x0efdc | DPIDR7        | 为SCS提供CoreSight发现信息                                       |
| 0x0efe0 | DPIDR0        | 为SCS提供CoreSight发现信息                                       |
| 0x0efe4 | DPIDR1        | 为SCS提供CoreSight发现信息                                       |
| 0x0efe8 | DPIDR2        | 为SCS提供CoreSight发现信息                                       |
| 0x0efec | DPIDR3        | 为SCS提供CoreSight发现信息                                       |
| 0x0eff0 | DCIDR0        | 为SCS提供CoreSight发现信息                                       |
| 0x0eff4 | DCIDR1        | 为SCS提供CoreSight发现信息                                       |
| 0x0eff8 | DCIDR2        | 为SCS提供CoreSight发现信息                                       |
| 0x0effc | DCIDR3        | 为SCS提供CoreSight发现信息                                       |
| 0x41004 | TRCPRGCTRLR   | 编程控制寄存器                                                   |
| 0x4100c | TRCSTATR      | TRCSTATR 指示 ETM-Teal 状态                                   |
| 0x41010 | TRCCONFIGR    | TRCCONFIGR 设置跟踪单元的基本跟踪选项                                  |
| 0x41020 | TRCEVENTCTL0R | TRCEVENTCTL0R 控制跟踪流中的事件跟踪。这些事件还驱动 ETM-Teal 的外部输出。         |
| 0x41024 | TRCEVENTCTL1R | TRCEVENTCTL1R 控制 TRCEVENTCTL0R 所选事件的行为                    |
| 0x4102c | TRCSTALLCTRLR | TRCSTALLCTRLR 使能 ETM-Teal 在 FIFO 超过设定阈值时暂停处理器，以降低溢出风险     |
| 0x41030 | TRCTSCTRLR    | TRCTSCTRLR 控制全局时间戳的插入，以导入跟踪流中。时间戳始终插入到指令跟踪流中。             |
| 0x41034 | TRCSYNCPR     | TRCSYNCPR 指定跟踪流的同步周期。TRCSYNCPR 定义请求跟踪同步之间的跟踪字节数。该值始终为二的幂。 |
| 0x41038 | TRCCCCTRLR    | TRCCCCTRLR 设定指令跟踪周期计数的阈值。该阈值表示周期计数跟踪包之间的最小间隔。             |
| 0x41080 | TRCVICTLR     | TRCVICTLR 控制指令跟踪过滤。                                       |
| 0x41140 | TRCCNTRLDVR0  | TRCCNTRLDVR 定义了简化功能计数器的重载值。                               |
| 0x41180 | TRCIDR8       | TRCIDR8                                                   |
| 0x41184 | TRCIDR9       | TRCIDR9                                                   |
| 0x41188 | TRCIDR10      | TRCIDR10                                                  |

| 偏移量     | 名称            | 说明                                        |
|---------|---------------|-------------------------------------------|
| 0x4118c | TRCIDR11      | TRCIDR11                                  |
| 0x41190 | TRCIDR12      | TRCIDR12                                  |
| 0x41194 | TRCIDR13      | TRCIDR13                                  |
| 0x411c0 | TRCIMSPEC     | TRCIMSPEC显示任何特定实现功能的存在，并启用所有提供的功能。        |
| 0x411e0 | TRCIDR0       | TRCIDR0                                   |
| 0x411e4 | TRCIDR1       | TRCIDR1                                   |
| 0x411e8 | TRCIDR2       | TRCIDR2                                   |
| 0x411ec | TRCIDR3       | TRCIDR3                                   |
| 0x411f0 | TRCIDR4       | TRCIDR4                                   |
| 0x411f4 | TRCIDR5       | TRCIDR5                                   |
| 0x411f8 | TRCIDR6       | TRCIDR6                                   |
| 0x411fc | TRCIDR7       | TRCIDR7                                   |
| 0x41208 | TRCRSCTLR2    | TRCRSCTLR 控制跟踪资源                          |
| 0x4120c | TRCRSCTLR3    | TRCRSCTLR 控制跟踪资源                          |
| 0x412a0 | TRCSSCSR      | 控制对应的单次比较器资源                              |
| 0x412c0 | TRCSSPCICR    | 选择单次控制用的PE比较器输入                           |
| 0x41310 | TRCPDCR       | 请求系统为追踪单元供电                               |
| 0x41314 | TRCPDSR       | 返回关于追踪单元的以下信息：- 操作系统锁定状态- 核心电源域状态- 电源中断状态 |
| 0x41ee4 | TRCITATBIDR   | 追踪集成ATB识别寄存器                              |
| 0x41ef4 | TRCITIATBINR  | 追踪集成指令ATB输入寄存器                            |
| 0x41efc | TRCITIATBOUTR | 追踪集成指令ATB输出寄存器                            |
| 0x41fa0 | TRCCLAIMSET   | 声明标签设置寄存器                                 |
| 0x41fa4 | TRCCLAIMCLR   | 声明标签清除寄存器                                 |
| 0x41fb8 | TRCAUTHSTATUS | 返回追踪单元所支持的跟踪级别                            |
| 0x41fbc | TRCDEVARCH    | TRCDEVARCH                                |
| 0x41fc8 | TRCDEVID      | TRCDEVID                                  |
| 0x41fcc | TRCDEVTYPE    | TRCDEVTYPE                                |
| 0x41fd0 | TRCPIDR4      | TRCPIDR4                                  |
| 0x41fd4 | TRCPIDR5      | TRCPIDR5                                  |
| 0x41fd8 | TRCPIDR6      | TRCPIDR6                                  |
| 0x41fdc | TRCPIDR7      | TRCPIDR7                                  |
| 0x41fe0 | TRCPIDR0      | TRCPIDR0                                  |
| 0x41fe4 | TRCPIDR1      | TRCPIDR1                                  |
| 0x41fe8 | TRCPIDR2      | TRCPIDR2                                  |

| 偏移量     | 名称               | 说明             |
|---------|------------------|----------------|
| 0x41fec | TRCPIDR3         | TRCPIDR3       |
| 0x41ff0 | TRCCIDR0         | TRCCIDR0       |
| 0x41ff4 | TRCCIDR1         | TRCCIDR1       |
| 0x41ff8 | TRCCIDR2         | TRCCIDR2       |
| 0x41ffc | TRCCIDR3         | TRCCIDR3       |
| 0x42000 | CTICONTROL       | CTI控制寄存器       |
| 0x42010 | CTIINTACK        | CTI 中断确认寄存器    |
| 0x42014 | CTIAPPSET        | CTI 应用触发设置寄存器  |
| 0x42018 | CTIAPPCLEAR      | CTI 应用触发清除寄存器  |
| 0x4201c | CTIAPPPULSE      | CTI 应用脉冲寄存器    |
| 0x42020 | CTIINEN0         | CTI触发至通道使能寄存器  |
| 0x42024 | CTIINEN1         | CTI触发至通道使能寄存器  |
| 0x42028 | CTIINEN2         | CTI触发至通道使能寄存器  |
| 0x4202c | CTIINEN3         | CTI触发至通道使能寄存器  |
| 0x42030 | CTIINEN4         | CTI触发至通道使能寄存器  |
| 0x42034 | CTIINEN5         | CTI触发至通道使能寄存器  |
| 0x42038 | CTIINEN6         | CTI触发至通道使能寄存器  |
| 0x4203c | CTIINEN7         | CTI触发至通道使能寄存器  |
| 0x420a0 | CTIOUTENO        | CTI触发至通道使能寄存器  |
| 0x420a4 | CTIOUTEN1        | CTI触发至通道使能寄存器  |
| 0x420a8 | CTIOUTEN2        | CTI触发至通道使能寄存器  |
| 0x420ac | CTIOUTEN3        | CTI触发至通道使能寄存器  |
| 0x420b0 | CTIOUTEN4        | CTI触发至通道使能寄存器  |
| 0x420b4 | CTIOUTEN5        | CTI触发至通道使能寄存器  |
| 0x420b8 | CTIOUTEN6        | CTI触发至通道使能寄存器  |
| 0x420bc | CTIOUTEN7        | CTI触发至通道使能寄存器  |
| 0x42130 | CTITRIGINSTATUS  | CTI触发至通道使能寄存器  |
| 0x42134 | CTITRIGOUTSTATUS | CTI 输入触发状态寄存器  |
| 0x42138 | CTICHINSTATUS    | CTI 通道输入状态寄存器  |
| 0x42140 | CTIGATE          | 使能 CTI 通道门控寄存器 |
| 0x42144 | ASICCTL          | 外部多路复用控制寄存器    |
| 0x42ee4 | ITCHOUT          | 集成测试通道输出寄存器    |
| 0x42ee8 | ITTRIGOUT        | 集成测试触发输出寄存器    |
| 0x42ef4 | ITCHIN           | 集成测试通道输入寄存器    |
| 0x42f00 | ITCTRL           | 集成模式控制寄存器      |
| 0x42fbc | DEVARCH          | 设备架构寄存器        |

| 偏移量     | 名称      | 说明               |
|---------|---------|------------------|
| 0x42fc8 | DEVID   | 设备配置寄存器          |
| 0x42fcc | DEVTYPE | 设备类型标识寄存器        |
| 0x42fd0 | PIDR4   | CoreSight 外设 ID4 |
| 0x42fd4 | PIDR5   | CoreSight 外设 ID5 |
| 0x42fd8 | PIDR6   | CoreSight 外设 ID6 |
| 0x42fdc | PIDR7   | CoreSight 外设 ID7 |
| 0x42fe0 | PIDR0   | CoreSight 外设 ID0 |
| 0x42fe4 | PIDR1   | CoreSight 外设 ID1 |
| 0x42fe8 | PIDR2   | CoreSight 外设 ID2 |
| 0x42fec | PIDR3   | CoreSight 外设 ID3 |
| 0x42ff0 | CIDR0   | CoreSight 组件 ID0 |
| 0x42ff4 | CIDR1   | CoreSight 组件 ID1 |
| 0x42ff8 | CIDR2   | CoreSight 组件 ID2 |
| 0x42ffc | CIDR3   | CoreSight 组件 ID3 |

## M33: ITM\_STIM0、ITM\_STIM1、...、ITM\_STIM30、ITM\_STIM31 寄存器

偏移: 0x00000、0x00004、...、0x00078、0x0007c

### 描述

提供生成仪表包的接口

表 122.  
ITM\_STIM0,  
ITM\_STIM1, ...  
ITM\_STIM30,  
ITM\_STIM31 寄存器

| 位    | 描述                                                           | 类型 | 复位         |
|------|--------------------------------------------------------------|----|------------|
| 31:0 | <b>STIMULUS:</b> 写入刺激端口 FIFO 的数据，用于转发为仪表包。写入访问的大小决定生成的仪表包类型。 | 读写 | 0x00000000 |

## M33: ITM\_TER0 寄存器

偏移: 0x00e00

### 说明

为每个 ITM\_STIM 寄存器配置独立的使能位

表 123. ITM\_TER0  
寄存器

| 位    | 描述                                                                   | 类型 | 复位         |
|------|----------------------------------------------------------------------|----|------------|
| 31:0 | <b>STIMENA:</b> 控制 ITM_TER*n 中的 STIMENA[m]，决定是否启用 ITM_STIM(32*n + m) | 读写 | 0x00000000 |

## M33: ITM\_TPR 寄存器

偏移: 0x00e40

### 描述

控制非特权代码可访问的刺激端口

表 124. ITM\_TPR  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:4 | 保留。 | -  | -  |

| 位   | 描述                                  | 类型 | 复位  |
|-----|-------------------------------------|----|-----|
| 3:0 | <b>PRIVMASK</b> : 启用 ITM 刺激端口跟踪的位掩码 | 读写 | 0x0 |

## M33: ITM\_TCR 寄存器

偏移: 0x00e80

### 描述

配置并控制通过ITM接口的传输

表 125. *ITM\_TCR* 寄存器

| 位     | 描述                                                           | 类型 | 复位   |
|-------|--------------------------------------------------------------|----|------|
| 31:24 | 保留。                                                          | -  | -    |
| 23    | <b>BUSY</b> : 指示 ITM 当前是否正在处理事件                              | 只读 | 0x0  |
| 22:16 | <b>TRACEBUSID</b> : 多源跟踪流格式标识符若启用多源跟踪，调试器必须向此字段写入唯一且非零的跟踪 ID | 读写 | 0x00 |
| 15:12 | 保留。                                                          | -  | -    |
| 11:10 | <b>GTSFREQ</b> : 定义 ITM 生成全局时间戳的频率，基于全局时间戳时钟频率，或禁用全局时间戳生成    | 读写 | 0x0  |
| 9:8   | <b>TSPRESCALE</b> : 本地时间戳预分频器，用于配合跟踪数据包参考时钟                  | 读写 | 0x0  |
| 7:6   | 保留。                                                          | -  | -    |
| 5     | <b>STALLENA</b> : 暂停处理引擎，以保证数据跟踪数据包的传输                       | 读写 | 0x0  |
| 4     | <b>SWOENA</b> : 启用时间戳计数器的异步时钟                                | 读写 | 0x0  |
| 3     | <b>TXENA</b> : 启用将硬件事件数据包从DWT单元转发至ITM，以供TPIU输出               | 读写 | 0x0  |
| 2     | <b>SYNCENA</b> : 启用同步TPIU的同步数据包传输                            | 读写 | 0x0  |
| 1     | <b>TSENA</b> : 启用本地时间戳生成                                     | 读写 | 0x0  |
| 0     | <b>ITMENA</b> : 启用ITM                                        | 读写 | 0x0  |

## M33: INT\_ATREADY寄存器

偏移量: 0x00ef0

### 说明

集成模式：读取ATB就绪信号

表 126.  
*INT\_ATREADY* 寄存器

| 位    | 描述                               | 类型 | 复位  |
|------|----------------------------------|----|-----|
| 31:2 | 保留。                              | -  | -   |
| 1    | <b>AFVALID</b> : 读取此位返回AFVALID的值 | 只读 | 0x0 |
| 0    | <b>ATREADY</b> : 读取此位返回ATREADY的值 | 只读 | 0x0 |

## M33: INT\_ATVALID寄存器

偏移量: 0x00ef8

**描述**

集成模式：写入ATB有效信号

表127。  
INT\_ATVALID寄存器

| 位    | 描述                                 | 类型 | 复位  |
|------|------------------------------------|----|-----|
| 31:2 | 保留。                                | -  | -   |
| 1    | <b>AFREADY</b> ：写入此位以设置AFREADY的值   | 读写 | 0x0 |
| 0    | <b>ATREADY</b> ：向该位写入即获得ATVALID 的值 | 读写 | 0x0 |

**M33：ITM\_ITCTRL 寄存器**

偏移：0x00f00

**描述**

集成模式控制寄存器

表128。  
ITM\_ITCTRL 寄存器

| 位    | 描述                                                                                       | 类型 | 复位  |
|------|------------------------------------------------------------------------------------------|----|-----|
| 31:1 | 保留。                                                                                      | -  | -   |
| 0    | <b>IME</b> ：集成模式使能位，可能取值为：0 - 跟踪单元不处于集成模式；1 - 跟踪单元处于集成模式。该模式使能：调试代理执行拓扑检测。SoC测试软件进行集成测试。 | 读写 | 0x0 |

**M33：ITM\_DEVARCH 寄存器**

偏移：0x00fbc

**描述**

提供ITM的CoreSight发现信息

表129。  
ITM\_DEVARCH  
寄存器

| 位     | 描述                                                                              | 类型 | 复位    |
|-------|---------------------------------------------------------------------------------|----|-------|
| 31:21 | <b>ARCHTECT</b> ：定义组件的架构商。位[31:28]为JEP106连续码（JEP106银行ID减1），位[27:21]为JEP106 ID码。 | 只读 | 0x23b |
| 20    | <b>PRESENT</b> ：定义DEVARC寄存器的存在                                                  | 只读 | 0x1   |
| 19:16 | <b>REVISION</b> ：定义组件的架构修订版                                                     | 只读 | 0x0   |
| 15:12 | <b>ARCHVER</b> ：定义组件的架构版本                                                       | 只读 | 0x1   |
| 11:0  | <b>ARCHPART</b> ：定义组件的架构                                                        | 只读 | 0xa01 |

**M33：ITM\_DEVTYPE寄存器**

偏移量：0x00fcc

**描述**

提供ITM的CoreSight发现信息

表130  
ITM\_DEVTYPE  
寄存器

| 位    | 描述                  | 类型 | 复位  |
|------|---------------------|----|-----|
| 31:8 | 保留。                 | -  | -   |
| 7:4  | <b>SUB</b> ：组件子类型   | 只读 | 0x4 |
| 3:0  | <b>MAJOR</b> ：组件主类型 | 只读 | 0x3 |

**M33：ITM\_PIDR4寄存器**

偏移量：0x00fd0

**描述**

提供ITM的CoreSight发现信息

表131. *ITM\_PIDR4*  
寄存器

| 位    | 描述                             | 类型 | 复位  |
|------|--------------------------------|----|-----|
| 31:8 | 保留。                            | -  | -   |
| 7:4  | <b>SIZE</b> : 详见CoreSight架构规范  | 只读 | 0x0 |
| 3:0  | <b>DES_2</b> : 详见CoreSight架构规范 | 只读 | 0x4 |

**M33：ITM\_PIDR5寄存器**

偏移量：0x00fd4

**描述**

提供ITM的CoreSight发现信息

表132. *ITM\_PIDR5*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33：ITM\_PIDR6寄存器**

偏移：0x00fd8

**描述**

提供ITM的CoreSight发现信息

表133. *ITM\_PIDR6*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33：ITM\_PIDR7寄存器**

偏移：0x00fdc

**描述**

提供ITM的CoreSight发现信息

表134. *ITM\_PIDR7*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33：ITM\_PIDR0寄存器**

偏移：0x00fe0

**描述**

提供ITM的CoreSight发现信息

表135. *ITM\_PIDR0*  
寄存器

| 位    | 描述                               | 类型 | 复位   |
|------|----------------------------------|----|------|
| 31:8 | 保留。                              | -  | -    |
| 7:0  | <b>PART_0:</b> 参见 CoreSight 架构规范 | 只读 | 0x21 |

## M33: ITM\_PIDR1 寄存器

偏移: 0x00fe4

### 描述

提供ITM的CoreSight发现信息

表136. *ITM\_PIDR1*  
寄存器

| 位    | 描述                               | 类型 | 复位  |
|------|----------------------------------|----|-----|
| 31:8 | 保留。                              | -  | -   |
| 7:4  | <b>DES_0:</b> 参见 CoreSight 架构规范  | 只读 | 0xb |
| 3:0  | <b>PART_1:</b> 参见 CoreSight 架构规范 | 只读 | 0xd |

## M33: ITM\_PIDR2 寄存器

偏移: 0x00fe8

### 说明

提供ITM的CoreSight发现信息

表137. *ITM\_PIDR2*  
寄存器

| 位    | 描述                            | 类型 | 复位  |
|------|-------------------------------|----|-----|
| 31:8 | 保留。                           | -  | -   |
| 7:4  | 版本: 详见CoreSight架构规范           | 只读 | 0x0 |
| 3    | <b>JEDEC:</b> 详见CoreSight架构规范 | 只读 | 0x1 |
| 2:0  | <b>DES_1:</b> 详见CoreSight架构规范 | 只读 | 0x3 |

## M33: ITM\_PIDR3 寄存器

偏移: 0x00fec

### 说明

提供ITM的CoreSight发现信息

表138. *ITM\_PIDR3*  
寄存器

| 位    | 描述                           | 类型 | 复位  |
|------|------------------------------|----|-----|
| 31:8 | 保留。                          | -  | -   |
| 7:4  | 修订: 详见CoreSight架构规范          | 只读 | 0x0 |
| 3:0  | <b>CMOD:</b> 详见CoreSight架构规范 | 只读 | 0x0 |

## M33: ITM\_CIDR0 寄存器

偏移: 0x00ff0

### 说明

提供ITM的CoreSight发现信息

表139. ITM\_CIDR0  
寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_0:</b> 详见 CoreSight 架构规范 | 只读 | 0x0d |

## M33: ITM\_CIDR1 寄存器

偏移: 0x00ff4

### 说明

提供ITM的CoreSight发现信息

表140. ITM\_CIDR1  
寄存器

| 位    | 描述                                | 类型 | 复位  |
|------|-----------------------------------|----|-----|
| 31:8 | 保留。                               | -  | -   |
| 7:4  | 类别: 参见 CoreSight 架构规范             | 只读 | 0x9 |
| 3:0  | <b>PRMBL_1:</b> 参见 CoreSight 架构规范 | 只读 | 0x0 |

## M33: ITM\_CIDR2 寄存器

偏移: 0x00ff8

### 说明

提供ITM的CoreSight发现信息

表141. ITM\_CIDR2  
寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_2:</b> 参见 CoreSight 架构规范 | 只读 | 0x05 |

## M33: ITM\_CIDR3 寄存器

偏移: 0x00ffc

### 说明

提供ITM的CoreSight发现信息

表142. ITM\_CIDR3  
寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_3:</b> 参见 CoreSight 架构规范 | 只读 | 0xb1 |

## M33: DWT\_CTRL 寄存器

偏移: 0x01000

### 说明

提供DWT单元的配置和状态信息，并用于控制该单元功能

表143. DWT\_CTRL  
寄存器

| 位     | 描述                              | 类型 | 复位  |
|-------|---------------------------------|----|-----|
| 31:28 | <b>NUMCOMP:</b> 已实现的 DWT 比较器数量  | 只读 | 0x7 |
| 27    | <b>NOTRCPKT:</b> 表示该实现不支持跟踪     | 只读 | 0x0 |
| 26    | <b>NOEXTTRIG:</b> 保留，置零保留 (RAZ) | 只读 | 0x0 |
| 25    | <b>NOCYCCNT:</b> 表示该实现不包含周期计数器  | 只读 | 0x1 |

| 位     | 描述                                                   | 类型 | 复位  |
|-------|------------------------------------------------------|----|-----|
| 24    | <b>NOPRFCNT</b> : 表示该实现不包含性能分析计数器                    | 只读 | 0x1 |
| 23    | <b>CYCDISS</b> : 控制在安全状态下是否禁用周期计数器                   | 读写 | 0x0 |
| 22    | <b>CYCEVTENA</b> : 使能POSTCNT下溢时事件计数包的生成              | 读写 | 0x1 |
| 21    | <b>FOLDEVTEA</b> : 使能DWT_FOLDCNT计数器                  | 读写 | 0x1 |
| 20    | <b>LSUEVTENA</b> : 使能DWT_LSU_CNT计数器                  | 读写 | 0x1 |
| 19    | <b>SLEEPEVTENA</b> : 使能DWT_SLEEP_CNT计数器              | 读写 | 0x0 |
| 18    | <b>EXCEVTENA</b> : 使能DWT_EXCCNT计数器                   | 读写 | 0x1 |
| 17    | <b>CPIEVTEA</b> : 使能DWT_CPI_CNT计数器                   | 读写 | 0x0 |
| 16    | <b>EXTTRCENA</b> : 使能异常跟踪包的生成                        | 读写 | 0x0 |
| 15:13 | 保留。                                                  | -  | -   |
| 12    | <b>PCSAMPLENA</b> : 使能使用POSTCNT计数器作为周期性PC采样包生成的定时器   | 读写 | 0x1 |
| 11:10 | <b>SYNCTAP</b> : 选择CYCCNT计数器上同步包计数器抽头的位置。该参数决定同步包的速率 | 读写 | 0x2 |
| 9     | <b>CYCTAP</b> : 选择CYCCNT计数器上POSTCNT抽头的位置。            | 读写 | 0x0 |
| 8:5   | <b>POSTINIT</b> : POSTCNT计数器的初始值。                    | 读写 | 0x1 |
| 4:1   | <b>POSTPRESET</b> : POSTCNT计数器的重装载值。                 | 读写 | 0x2 |
| 0     | <b>CYCCNTENA</b> : 启用CYCCNT。                         | 读写 | 0x0 |

## M33: DWT\_CYCCNT寄存器

偏移: 0x01004

### 描述

显示或设置处理器周期计数器CYCCNT的数值

表144。  
DWT\_CYCCNT  
寄存器

| 位    | 描述                                                                                            | 类型 | 复位         |
|------|-----------------------------------------------------------------------------------------------|----|------------|
| 31:0 | <b>CYCCNT</b> : 当DWT_CTRL.CYCCNTENA == 1且DEMCR.TRCENA == 1时, 每个处理器时钟周期递增1。溢出时, CYCCNT计数器回绕至零。 | 读写 | 0x00000000 |

## M33: DWT\_EXCCNT寄存器

偏移: 0x0100c

### 描述

统计异常处理期间消耗的总周期数

表145。  
DWT\_EXCCNT  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:8 | 保留。 | -  | -  |

| 位   | 描述                                                                                                                                                                                                                                   | 类型 | 复位   |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 7:0 | <b>EXCCNT:</b> 当以下所有条件均满足时，每个周期计数递增1： - DWT_CTRL.RL.EXCEVTENA == 1 且 DEMCR.TRCENA == 1。 - 未执行任何指令，详见 DWT_CPICT。 - 正在执行异常进入或异常退出相关操作。 - SecureNoninvasiveDebugAllowed() == TRUE，或该操作的NS-Req设置为非安全且 NoninvasiveDebugAllowed() == TRUE。 | 读写 | 0x00 |

## M33: DWT\_LSUCNT 寄存器

偏移: 0x01014

### 描述

记录执行所有加载或存储指令所需的额外周期数

表146。  
DWT\_LSUCNT 寄存器

| 位    | 描述                                                                                                                                                                                                                                                       | 类型 | 复位   |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                                                                                                                                                                                      | -  | -    |
| 7:0  | <b>LSUCNT:</b> 当以下所有条件满足时，每周期计数加一： - DWT_CTRL.LSUEVTENA == 1 且 DEMCR.TRCENA == 1。 - 未执行任何指令，详见 DWT_CPICT。 - 未执行异常进入或异常退出操作，详见 DWT_EXCCNT。 - 正在执行加载-存储操作。 - SecureNoninvasiveDebugAllowed() == TRUE，或该操作的NS-Req设置为非安全且 NoninvasiveDebugAllowed() == TRUE。 | 读写 | 0x00 |

## M33: DWT\_FOLDCNT 寄存器

偏移: 0x01018

### 描述

记录执行所有加载或存储指令所需的额外周期数

表 147.  
DWT\_FOLDCNT  
寄存器

| 位    | 描述                                                                                                                                                                                                                             | 类型 | 复位   |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                                                                                                                                                            | -  | -    |
| 7:0  | <b>FOLDCNT:</b> 当下列所有条件均满足时，每个周期累计计数： - DWT_CTRL.FOLDEVTENA == 1 且 DEMCR.TRCENA == 1。 - 至少执行两条指令，详见 DWT_CPICT。 - 或者 SecureNoninvasiveDebugAllowed() == TRUE，或处理引擎（PE）处于非安全态且 NoninvasiveDebugAllowed() == TRUE。计数器递增值为执行指令数减一。 | 读写 | 0x00 |

## M33: DWT\_COMPO 寄存器

偏移: 0x01020

表 148.  
DWT\_COMPO 寄存器

| 位    | 描述            | 类型 | 复位         |
|------|---------------|----|------------|
| 31:0 | 为观察点比较器0提供参考值 | 读写 | 0x00000000 |

## M33: DWT\_FUNCTION0 寄存器

偏移: 0x01028

### 描述

控制观察点比较器0的运行

表 149。  
DWT\_FUNCTION0  
寄存器

| 位     | 描述                                                          | 类型 | 复位   |
|-------|-------------------------------------------------------------|----|------|
| 31:27 | <b>ID:</b> 标识比较器 *n 匹配 (MATCH) 功能的能力                        | 只读 | 0x0b |
| 26:25 | 保留。                                                         | -  | -    |
| 24    | <b>MATCHED:</b> 当比较器匹配时置位为 1                                | 只读 | 0x0  |
| 23:12 | 保留。                                                         | -  | -    |
| 11:10 | <b>DATAVSIZE:</b> 定义由数据值和数据地址比较器监视的对象大小                     | 读写 | 0x0  |
| 9:6   | 保留。                                                         | -  | -    |
| 5:4   | <b>ACTION:</b> 定义匹配时执行的操作。如果通过MATCH禁用，则此字段将被忽略，且比较器不会生成任何操作 | 读写 | 0x0  |
| 3:0   | <b>MATCH:</b> 控制此比较器生成的匹配类型。                                | 读写 | 0x0  |

## M33: DWT\_COMP1 寄存器

偏移: 0x01030

表150。  
DWT\_COMP1寄存器

| 位    | 描述            | 类型 | 复位         |
|------|---------------|----|------------|
| 31:0 | 为监视点比较器1提供参考值 | 读写 | 0x00000000 |

## M33: DWT\_FUNCTION1 寄存器

偏移: 0x01038

### 描述

控制监视点比较器1的操作

表151。  
DWT\_FUNCTION1  
寄存器

| 位     | 描述                                                          | 类型 | 复位   |
|-------|-------------------------------------------------------------|----|------|
| 31:27 | <b>ID:</b> 标识比较器 *n 匹配 (MATCH) 功能的能力                        | 只读 | 0x11 |
| 26:25 | 保留。                                                         | -  | -    |
| 24    | <b>MATCHED:</b> 当比较器匹配时置位为 1                                | 只读 | 0x1  |
| 23:12 | 保留。                                                         | -  | -    |
| 11:10 | <b>DATAVSIZE:</b> 定义由数据值和数据地址比较器监视的对象大小                     | 读写 | 0x2  |
| 9:6   | 保留。                                                         | -  | -    |
| 5:4   | <b>ACTION:</b> 定义匹配时执行的操作。如果通过MATCH禁用，则此字段将被忽略，且比较器不会生成任何操作 | 读写 | 0x2  |
| 3:0   | <b>MATCH:</b> 控制此比较器生成的匹配类型。                                | 读写 | 0x8  |

## M33: DWT\_COMP2 寄存器

偏移: 0x01040

表152。  
DWT\_COMP2寄存器

| 位    | 描述            | 类型 | 复位         |
|------|---------------|----|------------|
| 31:0 | 为监视点比较器2提供参考值 | 读写 | 0x00000000 |

## M33: DWT\_FUNCTION2寄存器

偏移: 0x01048

### 描述

控制监视点比较器2的操作

表153。  
DWT\_FUNCTION2  
寄存器

| 位     | 描述                                                          | 类型 | 复位   |
|-------|-------------------------------------------------------------|----|------|
| 31:27 | <b>ID:</b> 标识比较器 *n 匹配 (MATCH) 功能的能力                        | 只读 | 0x0a |
| 26:25 | 保留。                                                         | -  | -    |
| 24    | <b>MATCHED:</b> 当比较器匹配时置位为 1                                | 只读 | 0x0  |
| 23:12 | 保留。                                                         | -  | -    |
| 11:10 | <b>DATAVSIZE:</b> 定义由数据值和数据地址比较器监视的对象大小                     | 读写 | 0x0  |
| 9:6   | 保留。                                                         | -  | -    |
| 5:4   | <b>ACTION:</b> 定义匹配时执行的操作。如果通过MATCH禁用，则此字段将被忽略，且比较器不会生成任何操作 | 读写 | 0x0  |
| 3:0   | <b>MATCH:</b> 控制此比较器生成的匹配类型。                                | 读写 | 0x0  |

## M33: DWT\_COMP3寄存器

偏移: 0x01050

表154。  
DWT\_COMP3寄存器

| 位    | 描述            | 类型 | 复位         |
|------|---------------|----|------------|
| 31:0 | 为监视点比较器3提供参考值 | 读写 | 0x00000000 |

## M33: DWT\_FUNCTION3寄存器

偏移: 0x01058

### 描述

控制监视点比较器3的操作

表155。  
DWT\_FUNCTION3  
寄存器

| 位     | 描述                                                          | 类型 | 复位   |
|-------|-------------------------------------------------------------|----|------|
| 31:27 | <b>ID:</b> 标识比较器 *n 匹配 (MATCH) 功能的能力                        | 只读 | 0x04 |
| 26:25 | 保留。                                                         | -  | -    |
| 24    | <b>MATCHED:</b> 当比较器匹配时置位为 1                                | 只读 | 0x0  |
| 23:12 | 保留。                                                         | -  | -    |
| 11:10 | <b>DATAVSIZE:</b> 定义由数据值和数据地址比较器监视的对象大小                     | 读写 | 0x2  |
| 9:6   | 保留。                                                         | -  | -    |
| 5:4   | <b>ACTION:</b> 定义匹配时执行的操作。如果通过MATCH禁用，则此字段将被忽略，且比较器不会生成任何操作 | 读写 | 0x0  |
| 3:0   | <b>MATCH:</b> 控制此比较器生成的匹配类型。                                | 读写 | 0x0  |

## M33: DWT\_DEVARCH 寄存器

偏移: 0x01fbc

### 描述

提供DWT的CoreSight发现信息

表156。  
DWT\_DEVARCH  
寄存器

| 位     | 描述                                                                               | 类型 | 复位    |
|-------|----------------------------------------------------------------------------------|----|-------|
| 31:21 | <b>ARCHTECT:</b> 定义组件的架构商。位[31:28]为JEP106连续码 (JEP106银行ID减1)，位[27:21]为JEP106 ID码。 | 只读 | 0x23b |
| 20    | <b>PRESENT:</b> 定义DEVARC寄存器的存在                                                   | 只读 | 0x1   |
| 19:16 | <b>REVISION:</b> 定义组件的架构修订版                                                      | 只读 | 0x0   |
| 15:12 | <b>ARCHVER:</b> 定义组件的架构版本                                                        | 只读 | 0x1   |
| 11:0  | <b>ARCHPART:</b> 定义组件的架构                                                         | 只读 | 0xa02 |

## M33: DWT\_DEVTYPEN 寄存器

偏移: 0x01fcc

### 说明

提供DWT的CoreSight发现信息

表157。  
DWT\_DEVTYPEN  
寄存器

| 位    | 描述                  | 类型 | 复位  |
|------|---------------------|----|-----|
| 31:8 | 保留。                 | -  | -   |
| 7:4  | <b>SUB:</b> 组件子类型   | 只读 | 0x0 |
| 3:0  | <b>MAJOR:</b> 组件主类型 | 只读 | 0x0 |

## M33: DWT\_PIDR4 寄存器

偏移: 0x01fd0

### 说明

提供DWT的CoreSight发现信息

表158。  
DWT\_PIDR4  
寄存器

| 位    | 描述                            | 类型 | 复位  |
|------|-------------------------------|----|-----|
| 31:8 | 保留。                           | -  | -   |
| 7:4  | <b>SIZE:</b> 详见CoreSight架构规范  | 只读 | 0x0 |
| 3:0  | <b>DES_2:</b> 详见CoreSight架构规范 | 只读 | 0x4 |

## M33: DWT\_PIDR5 寄存器

偏移: 0x01fd4

### 说明

提供DWT的CoreSight发现信息

表 159  
DWT\_PIDR5 寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: DWT\_PIDR6 寄存器

偏移: 0x01fd8

### 说明

提供DWT的CoreSight发现信息

表 160  
DWT\_PIDR6 寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: DWT\_PIDR7 寄存器

偏移: 0x01fdc

### 描述

提供DWT的CoreSight发现信息

表 161。  
DWT\_PIDR7 寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: DWT\_PIDR0 寄存器

偏移: 0x01fe0

### 描述

提供DWT的CoreSight发现信息

表 162。  
DWT\_PIDR0 寄存器

| 位    | 描述                               | 类型 | 复位   |
|------|----------------------------------|----|------|
| 31:8 | 保留。                              | -  | -    |
| 7:0  | <b>PART_0:</b> 参见 CoreSight 架构规范 | 只读 | 0x21 |

## M33: DWT\_PIDR1 寄存器

偏移: 0x01fe4

### 描述

提供DWT的CoreSight发现信息

表 163。  
DWT\_PIDR1 寄存器

| 位    | 描述                               | 类型 | 复位  |
|------|----------------------------------|----|-----|
| 31:8 | 保留。                              | -  | -   |
| 7:4  | <b>DES_0:</b> 参见 CoreSight 架构规范  | 只读 | 0xb |
| 3:0  | <b>PART_1:</b> 参见 CoreSight 架构规范 | 只读 | 0xd |

## M33: DWT\_PIDR2 寄存器

偏移: 0x01fe8

### 描述

提供DWT的CoreSight发现信息

表164。  
DWT\_PIDR2寄存器

| 位    | 描述                            | 类型 | 复位  |
|------|-------------------------------|----|-----|
| 31:8 | 保留。                           | -  | -   |
| 7:4  | 版本: 详见CoreSight架构规范           | 只读 | 0x0 |
| 3    | <b>JEDEC:</b> 详见CoreSight架构规范 | 只读 | 0x1 |
| 2:0  | <b>DES_1:</b> 详见CoreSight架构规范 | 只读 | 0x3 |

## M33: DWT\_PIDR3寄存器

偏移: 0x01fec

### 描述

提供DWT的CoreSight发现信息

表165。  
DWT\_PIDR3 寄存器

| 位    | 描述                           | 类型 | 复位  |
|------|------------------------------|----|-----|
| 31:8 | 保留。                          | -  | -   |
| 7:4  | 修订: 详见CoreSight架构规范          | 只读 | 0x0 |
| 3:0  | <b>CMOD:</b> 详见CoreSight架构规范 | 只读 | 0x0 |

## M33: DWT\_CIDR0 寄存器

偏移: 0x01ff0

### 描述

提供DWT的CoreSight发现信息

表 166。  
DWT\_CIDR0 寄存器

| 位    | 描述                              | 类型 | 复位   |
|------|---------------------------------|----|------|
| 31:8 | 保留。                             | -  | -    |
| 7:0  | <b>PRMBL_0:</b> 详见CoreSight架构规范 | 只读 | 0x0d |

## M33: DWT\_CIDR1 寄存器

偏移: 0x01ff4

### 描述

提供DWT的CoreSight发现信息

表 167。  
DWT\_CIDR1 寄存器

| 位    | 描述                                | 类型 | 复位  |
|------|-----------------------------------|----|-----|
| 31:8 | 保留。                               | -  | -   |
| 7:4  | 类别: 参见 CoreSight 架构规范             | 只读 | 0x9 |
| 3:0  | <b>PRMBL_1:</b> 参见 CoreSight 架构规范 | 只读 | 0x0 |

## M33: DWT\_CIDR2 寄存器

偏移: 0x01ff8

### 描述

提供DWT的CoreSight发现信息

表 168。  
DWT\_CIDR2 寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_2:</b> 参见 CoreSight 架构规范 | 只读 | 0x05 |

## M33: DWT\_CIDR3 寄存器

偏移: 0x01ffc

### 描述

提供DWT的CoreSight发现信息

表 169。  
DWT\_CIDR3 寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_3:</b> 参见 CoreSight 架构规范 | 只读 | 0xb1 |

## M33: FP\_CTRL 寄存器

偏移: 0x02000

### 描述

提供FPB实现信息及FPB单元的全局启用

表 170 FP\_CTRL 寄存器

| 位     | 描述                                                                               | 类型 | 复位  |
|-------|----------------------------------------------------------------------------------|----|-----|
| 31:28 | <b>REV:</b> Flash Patch 和断点单元架构版本                                                | 只读 | 0x6 |
| 27:15 | 保留。                                                                              | -  | -   |
| 14:12 | <b>NUM_CODE_14_12:</b> 指示已实现的指令地址比较器数量。值为零表示未实现指令地址比较器。指令地址比较器编号从0至NUM_CODE - 1。 | 只读 | 0x5 |
| 11:8  | <b>NUM_LIT:</b> 指示已实现的文字地址比较器数量。<br>文字地址比较器编号从NUM_CODE至NUM_CODE + NUM_LIT - 1。   | 只读 | 0x5 |
| 7:4   | <b>NUM_CODE_7_4:</b> 指示已实现的指令地址比较器数量。值为零表示未实现指令地址比较器。指令地址比较器编号从0至NUM_CODE - 1。   | 只读 | 0x8 |
| 3:2   | 保留。                                                                              | -  | -   |
| 1     | <b>KEY:</b> 除非同时写入KEY，否则对FP_CTRL的写操作将被忽略。                                        | 读写 | 0x0 |
| 0     | <b>ENABLE:</b> 启用FPB。                                                            | 读写 | 0x0 |

## M33: FP\_REMAP 寄存器

偏移: 0x02004

### 描述

指示实现是否支持Flash Patch重映射，若支持，则保存重映射目标地址

表 171 FP\_REMAP 寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:30 | 保留。 | -  | -  |

| 位    | 描述                                           | 类型 | 复位       |
|------|----------------------------------------------|----|----------|
| 29   | <b>RMPSPt</b> : 指示FPB单元是否支持Flash Patch重映射功能。 | 只读 | 0x0      |
| 28:5 | <b>REMAP</b> : 保存 Flash Patch 重映射地址的位[28:5]  | 只读 | 0x000000 |
| 4:0  | 保留。                                          | -  | -        |

## M33: FP\_COMP0、FP\_COMP1、...、FP\_COMP6、FP\_COMP7 寄存器

偏移量: 0x02008、0x0200c、...、0x02020、0x02024

### 描述

保存用于比较的地址。匹配结果取决于FPB配置及比较器为指令地址比较器或字面地址比较器

表172。FP\_COMP0、FP\_COMP1、...、FP\_COMP6、FP\_COMP7 寄存器

| 位    | 描述                               | 类型 | 复位  |
|------|----------------------------------|----|-----|
| 31:1 | 保留。                              | -  | -   |
| 0    | <b>BE</b> : 选择 Flash Patch 与断点功能 | 读写 | 0x0 |

## M33: FP\_DEVARCH 寄存器

偏移量: 0x02fbc

### 描述

提供FPB的CoreSight发现信息

表173。FP\_DEVARCH 寄存器

| 位     | 描述                                                                                 | 类型 | 复位    |
|-------|------------------------------------------------------------------------------------|----|-------|
| 31:21 | <b>ARCHITECT</b> : 定义组件的架构商。位[31:28]为JEP106连续码 (JEP106银行ID减1)，位[27:21]为JEP106 ID码。 | 只读 | 0x23b |
| 20    | <b>PRESENT</b> : 定义DEVARC寄存器的存在                                                    | 只读 | 0x1   |
| 19:16 | <b>REVISION</b> : 定义组件的架构修订版                                                       | 只读 | 0x0   |
| 15:12 | <b>ARCHVER</b> : 定义组件的架构版本                                                         | 只读 | 0x1   |
| 11:0  | <b>ARCHPART</b> : 定义组件的架构                                                          | 只读 | 0xa03 |

## M33: FP\_DEVTYPED 寄存器

偏移量: 0x02fcc

### 描述

提供FPB的CoreSight发现信息

表174。FP\_DEVTYPED 寄存器

| 位    | 描述                   | 类型 | 复位  |
|------|----------------------|----|-----|
| 31:8 | 保留。                  | -  | -   |
| 7:4  | <b>SUB</b> : 组件子类型   | 只读 | 0x0 |
| 3:0  | <b>MAJOR</b> : 组件主类型 | 只读 | 0x0 |

## M33: FP\_PIDR4 寄存器

偏移量: 0x02fd0

**描述**

提供FP的CoreSight发现信息

表175. FP\_PIDR4  
寄存器

| 位    | 描述                             | 类型 | 复位  |
|------|--------------------------------|----|-----|
| 31:8 | 保留。                            | -  | -   |
| 7:4  | <b>SIZE</b> : 详见CoreSight架构规范  | 只读 | 0x0 |
| 3:0  | <b>DES_2</b> : 详见CoreSight架构规范 | 只读 | 0x4 |

**M33: FP\_PIDR5寄存器**

偏移: 0x02fd4

**描述**

提供FP的CoreSight发现信息

表176 FP\_PIDR5寄  
存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33: FP\_PIDR6寄存器**

偏移: 0x02fd8

**描述**

提供FP的CoreSight发现信息

表177 FP\_PIDR6寄  
存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33: FP\_PIDR7寄存器**

偏移: 0x02fdc

**描述**

提供FP的CoreSight发现信息

表178. FP\_PIDR7寄  
存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33: FP\_PIDR0寄存器**

偏移: 0x02fe0

**描述**

提供FP的CoreSight发现信息

表179. FP\_PIDR0寄存器

| 位    | 描述                               | 类型 | 复位   |
|------|----------------------------------|----|------|
| 31:8 | 保留。                              | -  | -    |
| 7:0  | <b>PART_0:</b> 参见 CoreSight 架构规范 | 只读 | 0x21 |

## M33: FP\_PIDR1寄存器

偏移: 0x02fe4

### 描述

提供FP的CoreSight发现信息

表180. FP\_PIDR1寄存器

| 位    | 描述                               | 类型 | 复位  |
|------|----------------------------------|----|-----|
| 31:8 | 保留。                              | -  | -   |
| 7:4  | <b>DES_0:</b> 参见 CoreSight 架构规范  | 只读 | 0xb |
| 3:0  | <b>PART_1:</b> 参见 CoreSight 架构规范 | 只读 | 0xd |

## M33: FP\_PIDR2寄存器

偏移: 0x02fe8

### 描述

提供FP的CoreSight发现信息

表181. FP\_PIDR2寄存器

| 位    | 描述                            | 类型 | 复位  |
|------|-------------------------------|----|-----|
| 31:8 | 保留。                           | -  | -   |
| 7:4  | 版本: 详见CoreSight架构规范           | 只读 | 0x0 |
| 3    | <b>JEDEC:</b> 详见CoreSight架构规范 | 只读 | 0x1 |
| 2:0  | <b>DES_1:</b> 详见CoreSight架构规范 | 只读 | 0x3 |

## M33: FP\_PIDR3寄存器

偏移: 0x02fec

### 描述

提供FP的CoreSight发现信息

表182. FP\_PIDR3寄存器

| 位    | 描述                           | 类型 | 复位  |
|------|------------------------------|----|-----|
| 31:8 | 保留。                          | -  | -   |
| 7:4  | 修订: 详见CoreSight架构规范          | 只读 | 0x0 |
| 3:0  | <b>CMOD:</b> 详见CoreSight架构规范 | 只读 | 0x0 |

## M33: FP\_CIDR0寄存器

偏移: 0x02ff0

### 描述

提供FP的CoreSight发现信息

表183. FP\_CIDR0寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_0:</b> 详见 CoreSight 架构规范 | 只读 | 0x0d |

## M33: FP\_CIDR1寄存器

偏移: 0x02ff4

### 描述

提供FP的CoreSight发现信息

表184. FP\_CIDR1寄存器

| 位    | 描述                                | 类型 | 复位  |
|------|-----------------------------------|----|-----|
| 31:8 | 保留。                               | -  | -   |
| 7:4  | 类别: 参见 CoreSight 架构规范             | 只读 | 0x9 |
| 3:0  | <b>PRMBL_1:</b> 参见 CoreSight 架构规范 | 只读 | 0x0 |

## M33: FP\_CIDR2寄存器

偏移: 0x02ff8

### 描述

提供FP的CoreSight发现信息

表185. FP\_CIDR2寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_2:</b> 参见 CoreSight 架构规范 | 只读 | 0x05 |

## M33: FP\_CIDR3寄存器

偏移: 0x02ffc

### 描述

提供FP的CoreSight发现信息

表186. FP\_CIDR3寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_3:</b> 参见 CoreSight 架构规范 | 只读 | 0xb1 |

## M33: ICTR寄存器

偏移: 0x0e004

### 描述

提供中断控制器相关信息

表187. ICTR寄存器

| 位    | 描述                                                                                | 类型 | 复位  |
|------|-----------------------------------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                                               | -  | -   |
| 3:0  | <b>INTLINESNUM</b> : 指每个NVIC控制寄存器组中已实现的最高寄存器编号，或在NVIC_IPR*n的情况下，<br>4×INTLINESNUM | 只读 | 0x1 |

## M33: ACTLR寄存器

偏移量: 0x0e008

### 描述

提供实现定义的配置和控制选项

表188。ACTLR寄存器

| 位     | 描述                              | 类型 | 复位  |
|-------|---------------------------------|----|-----|
| 31:30 | 保留。                             | -  | -   |
| 29    | <b>EXTEXCLALL</b> : 无MPU时允许外部互斥 | 读写 | 0x0 |
| 28:13 | 保留。                             | -  | -   |
| 12    | <b>DISITMATBFLUSH</b> : 禁用ATB刷新 | 读写 | 0x0 |
| 11    | 保留。                             | -  | -   |
| 10    | <b>FPEXCODIS</b> : 禁用FPU异常输出    | 读写 | 0x0 |
| 9     | <b>DISOOFP</b> : 禁用乱序浮点指令完成     | 读写 | 0x0 |
| 8:3   | 保留。                             | -  | -   |
| 2     | <b>DISFOLD</b> : 禁用双发射          | 读写 | 0x0 |
| 1     | 保留。                             | -  | -   |
| 0     | <b>DISMCYCINT</b> : 禁用多周期中断     | 读写 | 0x0 |

## M33: SYST\_CSR寄存器

偏移量: 0x0e010

### 描述

使用SysTick控制和状态寄存器启用SysTick功能。

表189。SYST\_CSR寄存器

| 位     | 描述                                                                                                        | 类型 | 复位  |
|-------|-----------------------------------------------------------------------------------------------------------|----|-----|
| 31:17 | 保留。                                                                                                       | -  | -   |
| 16    | <b>COUNTFLAG</b> : 如果定时器自上次读取以来计数至0，返回1。<br>由应用程序或调试器在读取时清除。                                              | 只读 | 0x0 |
| 15:3  | 保留。                                                                                                       | -  | -   |
| 2     | <b>CLKSOURCE</b> : SysTick时钟源。若SYST_CALIB报告NOREF，则始终读作1。<br>选择SysTick定时器时钟源：<br>0 = 外部参考时钟。<br>1 = 处理器时钟。 | 读写 | 0x0 |
| 1     | <b>TICKINT</b> : 启用SysTick异常请求：<br>0 = 计数到零时不产生SysTick异常请求。<br>1 = 计数到零时产生SysTick异常请求。                    | 读写 | 0x0 |

| 位 | 描述                                                       | 类型 | 复位  |
|---|----------------------------------------------------------|----|-----|
| 0 | <b>ENABLE:</b> 启用SysTick计数器：<br>0 = 禁用计数器。<br>1 = 启用计数器。 | 读写 | 0x0 |

## M33: SYST\_RVR寄存器

偏移量: 0x0e014

### 说明

使用SysTick重载值寄存器指定计数器计数至0时加载入当前值寄存器的初始值。该初始值可为0至0xFFFFFFFF之间的任意值。起始值为0虽可行，但无效，因为SysTick中断和COUNTFLAG仅在从1计数至0时激活。该寄存器的复位值未知。

要生成一个周期为N个处理器时钟周期的多次定时器，应将RELOAD设为N-1。例如，若需每100个时钟脉冲触发一次SysTick中断，则将RELOAD设为99。

表190. SYST\_RVR  
寄存器

| 位     | 描述                                          | 类型 | 复位       |
|-------|---------------------------------------------|----|----------|
| 31:24 | 保留。                                         | -  | -        |
| 23:0  | <b>RELOAD:</b> 计数器计数到0时加载到SysTick当前值寄存器的数值。 | 读写 | 0x000000 |

## M33: SYST\_CVR寄存器

偏移地址: 0x0e018

### 描述

使用SysTick当前值寄存器以读取寄存器中的当前数值。该寄存器的复位值未知。

表191. SYST\_CVR  
寄存器

| 位     | 描述                                                                                                           | 类型 | 复位       |
|-------|--------------------------------------------------------------------------------------------------------------|----|----------|
| 31:24 | 保留。                                                                                                          | -  | -        |
| 23:0  | <b>CURRENT:</b> 读取操作返回SysTick计数器的当前数值。该寄存器为写入清零型。<br>向该寄存器写入任意值均可将其清零。清零该寄存器同时清除SysTick控制与状态寄存器中的COUNTFLAG位。 | 读写 | 0x000000 |

## M33: SYST\_CALIB寄存器

偏移: 0x0e01c

### 描述

使用 SysTick 校准值寄存器使软件能够通过除法和乘法调整至任意所需速度。

表192。  
SYST\_CALIB 寄存器

| 位     | 描述                                                                            | 类型 | 复位  |
|-------|-------------------------------------------------------------------------------|----|-----|
| 31    | <b>NOREF:</b> 若读数为 1，表示未提供参考时钟—SysTick 控制和状态寄存器中的 CL KSOURCE 位将被强制置为 1，且不可清零。 | 只读 | 0x0 |
| 30    | <b>SKEW:</b> 若读数为 1，表示 10ms 校准值存在偏差（因时钟频率所致）。                                 | 只读 | 0x0 |
| 29:24 | 保留。                                                                           | -  | -   |

| 位    | 描述                                                                 | 类型 | 复位       |
|------|--------------------------------------------------------------------|----|----------|
| 23:0 | <b>TENMS:</b> 用于 10ms (100Hz) 计时的可选重载值，受系统时钟偏差误差影响。若读数为 0，表示校准值未知。 | 只读 | 0x000000 |

## M33: NVIC\_ISERO, NVIC\_ISER1 寄存器

偏移: 0x0e100, 0x0e104

### 描述

启用或读取每组32个中断的启用状态

表 193。  
NVIC\_ISERO,  
NVIC\_ISER1 寄存器

| 位    | 描述                                                               | 类型 | 复位         |
|------|------------------------------------------------------------------|----|------------|
| 31:0 | <b>SETENA:</b> 针对 NVIC_ISER*n 中的 SETENA[m]，指示中断 $32*n + m$ 是否已启用 | 读写 | 0x00000000 |

## M33: NVIC\_ICERO、NVIC\_ICER1 寄存器

偏移量: 0x0e180、0x0e184

### 描述

清除或读取每组32个中断的启用状态

表 194。  
NVIC\_ICERO,  
NVIC\_ICER1 寄存器

| 位    | 描述                                                               | 类型 | 复位         |
|------|------------------------------------------------------------------|----|------------|
| 31:0 | <b>CLRENA:</b> 针对 NVIC_ICER*n 中的 CLRENA[m]，指示中断 $32*n + m$ 是否已启用 | 读写 | 0x00000000 |

## M33: NVIC\_ISPR0、NVIC\_ISPR1 寄存器

偏移量: 0x0e200、0x0e204

### 描述

启用或读取每组32个中断的挂起状态

表 195。  
NVIC\_ISPR0,  
NVIC\_ISPR1 寄存器

| 位    | 描述                                                                     | 类型 | 复位         |
|------|------------------------------------------------------------------------|----|------------|
| 31:0 | <b>SETPEND:</b> 针对 NVIC_ISPR*n 中的 SETPEND[m]，指示中断 $32*n + m$ 是否处于待处理状态 | 读写 | 0x00000000 |

## M33: NVIC\_ICPR0、NVIC\_ICPR1 寄存器

偏移量: 0x0e280、0x0e284

### 描述

清除或读取每组32个中断的挂起状态

表 196。  
NVIC\_ICPR0,  
NVIC\_ICPR1 寄存器

| 位    | 描述                                                              | 类型 | 复位         |
|------|-----------------------------------------------------------------|----|------------|
| 31:0 | <b>CLRPEND:</b> 在 NVIC_ICPR*n 中，CLRPEND[m] 表示中断 $32*n + m$ 是否挂起 | 读写 | 0x00000000 |

## M33: NVIC\_IABR0, NVIC\_IABR1 寄存器

偏移量: 0x0e300, 0x0e304

### 描述

针对每组32个中断，显示每个中断的活动状态

表197。  
NVIC\_IABR0,  
NVIC\_IABR1 寄存器

| 位    | 描述                                                            | 类型 | 复位         |
|------|---------------------------------------------------------------|----|------------|
| 31:0 | <b>ACTIVE:</b> 在 NVIC_IABR*n 中, ACTIVE[m] 表示中断的激活状态<br>32*n+m | 读写 | 0x00000000 |

## M33: NVIC\_ITNS0, NVIC\_ITNS1 寄存器

偏移量: 0x0e380, 0x0e384

### 描述

针对每组32个中断, 确定每个中断的目标是非安全态还是安全态

表198。  
NVIC\_ITNS0,  
NVIC\_ITNS1 寄存器

| 位    | 描述                                                          | 类型 | 复位         |
|------|-------------------------------------------------------------|----|------------|
| 31:0 | <b>ITNS:</b> 在 NVIC_ITNS*n 中, ITNS[m] 表示中断 32*n+m 所属的目标安全状态 | 读写 | 0x00000000 |

## M33: NVIC\_IPR0, NVIC\_IPR1, ..., NVIC\_IPR14, NVIC\_IPR15 寄存器

偏移量: 0x0e400, 0x0e404, ..., 0x0e438, 0x0e43c

### 描述

设置或读取中断优先级

表199。NVIC\_IPR0、N  
VIC\_IPR1、...  
、NVIC\_IPR  
14、NVIC\_IPR15寄存器

| 位     | 描述                                                             | 类型 | 复位  |
|-------|----------------------------------------------------------------|----|-----|
| 31:28 | <b>PRI_N3:</b> 对寄存器NVIC_IPRn, 中断号4*n+3的优先级, 或若PE未实现该中断, 则为RES0 | 读写 | 0x0 |
| 27:24 | 保留。                                                            | -  | -   |
| 23:20 | <b>PRI_N2:</b> 对寄存器NVIC_IPRn, 中断号4*n+2的优先级, 或若PE未实现该中断, 则为RES0 | 读写 | 0x0 |
| 19:16 | 保留。                                                            | -  | -   |
| 15:12 | <b>PRI_N1:</b> 对寄存器NVIC_IPRn, 中断号4*n+1的优先级, 或若PE未实现该中断, 则为RES0 | 读写 | 0x0 |
| 11:8  | 保留。                                                            | -  | -   |
| 7:4   | <b>PRI_N0:</b> 对寄存器NVIC_IPRn, 中断号4*n+0的优先级, 或若PE未实现该中断, 则为RES0 | 读写 | 0x0 |
| 3:0   | 保留。                                                            | -  | -   |

## M33: CPUID寄存器

偏移量: 0x0ed00

### 描述

提供PE的识别信息, 包括设备的实现者代码及设备ID编号

表200。CPUID  
寄存器

| 位     | 描述                                                     | 类型 | 复位   |
|-------|--------------------------------------------------------|----|------|
| 31:24 | <b>IMPLEMENTER:</b> 该字段必须包含由ARM分配的实施者代码                | 只读 | 0x41 |
| 23:20 | <b>VARIANT:</b> 实现自定义的变体编号。通常, 此字段用于区分不同的产品变体或产品的主要版本。 | 只读 | 0x1  |
| 19:16 | <b>ARCHITECTURE:</b> 定义PE所实现的架构。                       | 只读 | 0xf  |

| 位    | 描述                             | 类型 | 复位    |
|------|--------------------------------|----|-------|
| 15:4 | <b>PARTNO</b> : 设备的实现自定义主零件编号。 | 只读 | 0xd21 |
| 3:0  | <b>REVISION</b> : 设备的实现自定义修订号。 | 只读 | 0x0   |

## M33: ICSR寄存器

偏移量: 0x0ed04

### 描述

控制并提供 NMI、PendSV、SysTick 及中断的状态信息

表201. ICSR  
寄存器

| 位     | 描述                                              | 类型 | 复位    |
|-------|-------------------------------------------------|----|-------|
| 31    | <b>PENDNMISET</b> : 指示NMI异常是否处于挂起状态。            | 只读 | 0x0   |
| 30    | <b>PENDNMICLR</b> : 允许清除NMI异常的挂起状态。             | 读写 | 0x0   |
| 29    | 保留。                                             | -  | -     |
| 28    | <b>PENDSVSET</b> : 指示PendSV `FTSSS异常是否处于挂起状态。   | 只读 | 0x0   |
| 27    | <b>PENDSVCLR</b> : 允许清除PendSV异常的挂起状态`FTSSS。     | 读写 | 0x0   |
| 26    | <b>PENDSTSET</b> : 指示SysTick的`FTSSS异常是否处于挂起状态   | 只读 | 0x0   |
| 25    | <b>PENDSTCLR</b> : 允许清除SysTick异常的挂起状态 `FTSSS    | 读写 | 0x0   |
| 24    | <b>STTNS</b> : 控制单个SysTick实现中, SysTick为安全或非安全状态 | 读写 | 0x0   |
| 23    | <b>ISRPREEMPT</b> : 指示是否在调试暂停状态退出时处理待处理异常       | 只读 | 0x0   |
| 22    | <b>ISR PENDING</b> : 指示由NVIC产生的外部中断是否处于挂起状态     | 只读 | 0x0   |
| 21    | 保留。                                             | -  | -     |
| 20:12 | <b>VECTPENDING</b> : 优先级最高且已启用的挂起中断的异常号         | 只读 | 0x000 |
| 11    | <b>RETTOBASE</b> : 在处理程序模式下, 指示是否存在多个活动异常       | 只读 | 0x0   |
| 10:9  | 保留。                                             | -  | -     |
| 8:0   | <b>VECTACTIVE</b> : 当前正在执行的异常号                  | 只读 | 0x000 |

## M33: VTOR寄存器

Offset: 0x0ed08

### 描述

VTOR指示向量表基地址相对于内存地址0x的偏移量00000000.

表202. VTOR  
寄存器

| 位    | 描述                                                        | 类型 | 复位        |
|------|-----------------------------------------------------------|----|-----------|
| 31:7 | <b>TBLOFF</b> : 向量表基偏移字段。它包含表基址相对于内存映射底部的偏移量的 bits[31:7]。 | 读写 | 0x0000000 |
| 6:0  | 保留。                                                       | -  | -         |

## M33：AIRCR 寄存器

偏移: 0x0ed0c

### 描述

使用应用程序中断和复位控制寄存器以：确定数据字节序，从调试停止模式清除所有活动状态信息，发起系统复位请求。

表 203。AIRCR  
寄存器

| 位     | 描述                                                                                                                                                                                                                                                                                                                                                                                             | 类型 | 复位     |
|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|--------|
| 31:16 | <b>VECTKEY</b> : 寄存器密钥：<br>读作未知<br>写入时，必须向 VECTKEY 写入 0x05FA，否则写入操作将被忽略。                                                                                                                                                                                                                                                                                                                       | 读写 | 0x0000 |
| 15    | <b>ENDIANESS</b> : 实现的数据字节序：<br>0 = 小端序。                                                                                                                                                                                                                                                                                                                                                       | 只读 | 0x0    |
| 14    | <b>PRIS</b> : 优先处理安全异常。该位值决定是否启用安全异常的优先级提升。<br>0 表示安全异常与非安全异常优先级范围相同。<br>1 表示非安全异常优先级被降低。                                                                                                                                                                                                                                                                                                      | 读写 | 0x0    |
| 13    | <b>BFHFNMIN</b> : 允许总线故障、硬故障和 NMI 的非安全访问。<br>0 表示总线故障、硬故障和 NMI 均为安全访问。<br>1 BusFault 和 NMI 为非安全态，异常可定位至非安全态硬故障。                                                                                                                                                                                                                                                                                | 读写 | 0x0    |
| 12:11 | 保留。                                                                                                                                                                                                                                                                                                                                                                                            | -  | -      |
| 10:8  | <b>PRIGROUP</b> : 中断优先级分组字段。该字段用于确定组优先级与子优先级的划分。<br><a href="https://developer.arm.com/documentation/100235/0004/the-cortex-m33-peripherals/system-control-block/application-interrupt-and-reset-control-register?lang=en">详见 https://developer.arm.com/documentation/100235/0004/the-cortex-m33-peripherals/system-control-block/application-interrupt-and-reset-control-register?lang=en</a> | 读写 | 0x0    |
| 7:4   | 保留。                                                                                                                                                                                                                                                                                                                                                                                            | -  | -      |
| 3     | <b>SYSRESETREQS</b> : 系统复位请求，仅限安全态。<br>0 表示 SYSRESETREQ 功能对两个安全态均可用。<br>1 表示 SYSRESETREQ 功能仅对安全态可用。                                                                                                                                                                                                                                                                                            | 读写 | 0x0    |
| 2     | <b>SYSRESETREQ</b> : 写1至该位将触发外部系统的 SYSRESETREQ 信号以请求复位。其目的是强制对除调试外所有主要组件执行大规模系统复位。系统复位请求将导致 DHCSR 中的 C_HALT 位被清除。调试器不会与设备断开连接。                                                                                                                                                                                                                                                                 | 读写 | 0x0    |
| 1     | <b>VECTCLACTIVE</b> : 清除所有固定及可配置异常的活动状态信息。<br>该位为自清位，仅当核心处于暂停状态时由DAP设置。置位时：清除处理器所有活动异常状态，强制返回线程模式，强制将IPSR设为0。调试器必须重新初始化堆栈。                                                                                                                                                                                                                                                                     | 读写 | 0x0    |
| 0     | 保留。                                                                                                                                                                                                                                                                                                                                                                                            | -  | -      |

## M33：SCR寄存器

偏移量：0x0ed10

### 描述

系统控制寄存器。该寄存器用于电源管理功能：向系统发出信号，指示处理器可进入低功耗状态，并控制处理器如何进入及退出低功耗状态。

表204. SCR  
寄存器

| 位    | 描述                                                                                                                                                                                                              | 类型 | 复位  |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:5 | 保留。                                                                                                                                                                                                             | -  | -   |
| 4    | <b>SEVONPEND</b> : 挂起时发送事件位：<br>0 = 仅启用的中断或事件可唤醒处理器，禁用的中断除外。<br><br>1 = 启用的事件及所有中断（包含禁用中断）均可唤醒处理器。<br><br>当事件或中断变为挂起时，事件信号将唤醒处理器退出WFE。如果<br><br>处理器未处于等待事件状态，事件将被注册并影响下一次WFE。<br><br>处理器亦会在执行 SEV 指令或发生外部事件时唤醒。 | 读写 | 0x0 |
| 3    | <b>SLEEPDEEPS</b> : 0 表示 SLEEPDEEP 对两种安全状态均可用。<br>1 表示 SLEEPDEEP 仅对安全状态可用。                                                                                                                                      | 读写 | 0x0 |
| 2    | <b>SLEEPDEEP</b> : 控制处理器以睡眠还是深度睡眠作为低功耗模式：<br><br>0 = 睡眠。<br>1 = 深度睡眠。                                                                                                                                           | 读写 | 0x0 |
| 1    | <b>SLEEPONEXIT</b> : 指示从处理程序模式返回线程模式时是否进入睡眠状态：<br><br>0 = 返回线程模式时不进入睡眠。<br>1 = 从 ISR 返回线程模式时进入睡眠或深度睡眠。<br>将此位设为 1，可使中断驱动应用避免返回至空的主程序。                                                                           | 读写 | 0x0 |
| 0    | 保留。                                                                                                                                                                                                             | -  | -   |

## M33：CCR 寄存器

偏移：0x0ed14

### 描述

设置或返回配置与控制数据

表205. CCR寄  
存器

| 位     | 描述                                  | 类型 | 复位  |
|-------|-------------------------------------|----|-----|
| 31:19 | 保留。                                 | -  | -   |
| 18    | <b>BP</b> : 启用程序流程预测 FTSSS          | 只读 | 0x0 |
| 17    | <b>IC</b> : 所选安全状态下指令缓存的全局使能位       | 只读 | 0x0 |
| 16    | <b>DC</b> : 启用对所有访问普通内存的数据缓存 `FTSSS | 只读 | 0x0 |
| 15:11 | 保留。                                 | -  | -   |

| 位   | 描述                                                    | 类型 | 复位  |
|-----|-------------------------------------------------------|----|-----|
| 10  | <b>STKOFHFMIGN:</b> 控制在请求优先级小于0执行时栈限制违规的影响            | 读写 | 0x0 |
| 9   | <b>RES1:</b> 保留, RES1                                 | 只读 | 0x1 |
| 8   | <b>BFHFMIGN:</b> 决定精确BusFault对请求优先级小于0的处理程序的影响        | 读写 | 0x0 |
| 7:5 | 保留。                                                   | -  | -   |
| 4   | <b>DIV_0_TRP:</b> 控制尝试进行整数除零时是否生成DIVBYZERO UsageFault | 读写 | 0x0 |
| 3   | <b>UNALIGN_TRP:</b> 控制未对齐字或半字访问的陷阱行为                  | 读写 | 0x0 |
| 2   | 保留。                                                   | -  | -   |
| 1   | <b>USERSETMPEND:</b> 决定是否允许无特权访问通过STIR挂起中断            | 读写 | 0x0 |
| 0   | <b>RES1_1:</b> 保留, RES1                               | 只读 | 0x1 |

## M33: SHPR1 寄存器

偏移: 0xed18

### 说明

设置或返回系统处理程序 4 至 7 的优先级

表206. SHPR1  
寄存器

| 位     | 描述                                         | 类型 | 复位  |
|-------|--------------------------------------------|----|-----|
| 31:29 | <b>PRI_7_3:</b> 系统处理程序7 (SecureFault) 的优先级 | 读写 | 0x0 |
| 28:24 | 保留。                                        | -  | -   |
| 23:21 | <b>PRI_6_3:</b> 系统处理程序6的优先级, SecureFault   | 读写 | 0x0 |
| 20:16 | 保留。                                        | -  | -   |
| 15:13 | <b>PRI_5_3:</b> 系统处理程序5的优先级, SecureFault   | 读写 | 0x0 |
| 12:8  | 保留。                                        | -  | -   |
| 7:5   | <b>PRI_4_3:</b> 系统处理程序4的优先级, SecureFault   | 读写 | 0x0 |
| 4:0   | 保留。                                        | -  | -   |

## M33: SHPR2 寄存器

偏移: 0xed1c

### 描述

设置或返回系统处理程序 8 至 11 的优先级

表207. SHPR2  
寄存器

| 位     | 描述                                         | 类型 | 复位   |
|-------|--------------------------------------------|----|------|
| 31:29 | <b>PRI_11_3:</b> 系统处理程序11的优先级, SecureFault | 读写 | 0x0  |
| 28:24 | 保留。                                        | -  | -    |
| 23:16 | <b>PRI_10:</b> 保留, RES0                    | 只读 | 0x00 |
| 15:8  | <b>PRI_9:</b> 保留, RES0                     | 只读 | 0x00 |

| 位   | 描述                     | 类型 | 复位   |
|-----|------------------------|----|------|
| 7:0 | <b>PRI_8:</b> 保留, RES0 | 只读 | 0x00 |

## M33: SHPR3 寄存器

偏移: 0x0ed20

### 描述

设置或返回系统处理程序 12 至 15 的优先级

表208. SHPR3 寄存器

| 位     | 描述                                         | 类型 | 复位   |
|-------|--------------------------------------------|----|------|
| 31:29 | <b>PRI_15_3:</b> 系统处理程序15的优先级, SecureFault | 读写 | 0x0  |
| 28:24 | 保留。                                        | -  | -    |
| 23:21 | <b>PRI_14_3:</b> 系统处理程序14的优先级, SecureFault | 读写 | 0x0  |
| 20:16 | 保留。                                        | -  | -    |
| 15:8  | <b>PRI_13:</b> 保留, RES0                    | 只读 | 0x00 |
| 7:5   | <b>PRI_12_3:</b> 系统处理程序12, SecureFault的优先级 | 读写 | 0x0  |
| 4:0   | 保留。                                        | -  | -    |

## M33: SHCSR 寄存器

偏移量: 0x0ed24

### 说明

提供对系统异常活动和挂起状态的访问

表209. SHCSR 寄存器

| 位     | 描述                                                                            | 类型 | 复位  |
|-------|-------------------------------------------------------------------------------|----|-----|
| 31:22 | 保留。                                                                           | -  | -   |
| 21    | <b>HARDFAULTPENDED:</b> `IAAMO 硬故障异常的挂起状态`CTTSSS                              | 读写 | 0x0 |
| 20    | <b>SECUREFAULTPENDED:</b> `IAAMO 安全故障异常的挂起状态`                                 | 读写 | 0x0 |
| 19    | <b>SECUREFAULTENA:</b> `DW 启用安全故障异常`                                          | 读写 | 0x0 |
| 18    | <b>USGFAULTENA:</b> `DW 启用用法故障异常`FTSSS                                        | 读写 | 0x0 |
| 17    | <b>BUSFAULTENA:</b> `DW 启用总线故障异常`                                             | 读写 | 0x0 |
| 16    | <b>MEMFAULTENA:</b> `DW 启用存储管理异常`FTSSS                                        | 读写 | 0x0 |
| 15    | <b>SVCALLPENDED:</b> `IAAMO SVCall异常的挂起状态`FTSSS                               | 读写 | 0x0 |
| 14    | <b>BUSFAULTPENDED:</b> `IAAMO 总线故障异常的挂起状态`                                    | 读写 | 0x0 |
| 13    | <b>MEMFAULTPENDED:</b> `IAAMO MemManage异常的挂起状态`FTSSS                          | 读写 | 0x0 |
| 12    | <b>USGFAULTPENDED:</b> UsageFault异常在安全状态之间被挂起, `IAAMO UsageFault异常的挂起状态`FTSSS | 读写 | 0x0 |
| 11    | <b>SYSTICKACT:</b> `IAAMO SysTick异常的激活状态`FTSSS                                | 读写 | 0x0 |
| 10    | <b>PENDSVACT:</b> `IAAMO PendSV异常的激活状态`FTSSS                                  | 读写 | 0x0 |

| 位 | 描述                                                         | 类型 | 复位  |
|---|------------------------------------------------------------|----|-----|
| 9 | 保留。                                                        | -  | -   |
| 8 | <b>MONITORACT</b> : `IAAMO DebugMonitor 异常的激活状态            | 读写 | 0x0 |
| 7 | <b>SVCALLACT</b> : `IAAMO SVCall 异常的激活状态`FTSSS             | 读写 | 0x0 |
| 6 | 保留。                                                        | -  | -   |
| 5 | <b>NMIACT</b> : `IAAMO NMI 异常的激活状态                         | 读写 | 0x0 |
| 4 | <b>SECUREFAULTACT</b> : `IAAMO SecureFault 异常的激活状态         | 读写 | 0x0 |
| 3 | <b>USGFAULTACT</b> : `IAAMO UsageFault 异常的激活状态`FTSSS       | 读写 | 0x0 |
| 2 | <b>HARDFAULTACT</b> : 指示并允许对 HardFault 异常的活动状态进行有限修改`FTSSS | 读写 | 0x0 |
| 1 | <b>BUSFAULTACT</b> : `IAAMO 总线错误异常的活动状态                    | 读写 | 0x0 |
| 0 | <b>MEMFAULTACT</b> : `IAAMO MemManage 异常的活动状态`FTSSS        | 读写 | 0x0 |

## M33: CFSR 寄存器

偏移: 0x0ed28

### 描述

包含三个可配置故障状态寄存器

31:16 UFSR: 提供有关 UsageFault 异常的信息

15:8 BFRS: 提供有关 BusFault 异常的信息

7:0 MMFSR: 提供有关 MemManage 异常的信息

表 210. CFSR  
寄存器

| 位     | 描述                                                         | 类型 | 复位  |
|-------|------------------------------------------------------------|----|-----|
| 31:26 | 保留。                                                        | -  | -   |
| 25    | <b>UFSR_DIVBYZERO</b> : 粘滞标志, 指示是否发生整数除零错误                 | 读写 | 0x0 |
| 24    | <b>UFSR_UNALIGNED</b> : 粘滞标志, 指示是否发生非对齐访问错误                | 读写 | 0x0 |
| 23:21 | 保留。                                                        | -  | -   |
| 20    | <b>UFSR_STKOF</b> : 粘滞标志, 指示是否发生栈溢出错误                      | 读写 | 0x0 |
| 19    | <b>UFSR_NOCP</b> : 粘滞标志, 指示是否发生协处理器禁用或不存在错误                | 读写 | 0x0 |
| 18    | <b>UFSR_INVPC</b> : 粘滞标志, 指示是否发生完整性校验错误                    | 读写 | 0x0 |
| 17    | <b>UFSR_INVSTATE</b> : 粘滞标志, 指示是否发生 EPSR.T 或 EPSR.IT 有效性错误 | 读写 | 0x0 |
| 16    | <b>UFSR_UNDEFINSTR</b> : 指示是否发生未定义指令错误的粘滞标志                | 读写 | 0x0 |
| 15    | <b>BFRS_BFARVALID</b> : 指示BFAR寄存器内容的有效性                    | 读写 | 0x0 |
| 14    | 保留。                                                        | -  | -   |

| 位   | 描述                                             | 类型 | 复位   |
|-----|------------------------------------------------|----|------|
| 13  | <b>BFSR_LSPERR</b> : 记录是否在浮点延迟状态保存期间发生总线错误     | 读写 | 0x0  |
| 12  | <b>BFSR_STKERR</b> : 记录是否在异常进入堆栈操作期间发生派生总线错误   | 读写 | 0x0  |
| 11  | <b>BFSR_UNSTKERR</b> : 记录是否在异常返回出栈操作期间发生派生总线错误 | 读写 | 0x0  |
| 10  | <b>BFSR_IMPRECISERR</b> : 记录是否发生非精确数据访问错误      | 读写 | 0x0  |
| 9   | <b>BFSR_PRECISERR</b> : 记录是否发生精确数据访问错误         | 读写 | 0x0  |
| 8   | <b>BFSR_IBUSERR</b> : 记录是否发生指令预取时的总线错误         | 读写 | 0x0  |
| 7:0 | <b>MMFSR</b> : 提供内存管理异常信息                      | 读写 | 0x00 |

## M33: HFSR 寄存器

偏移: 0x0ed2c

### 描述

显示任何 HardFault 的原因

表211. HFSR  
寄存器

| 位    | 描述                                                                 | 类型 | 复位  |
|------|--------------------------------------------------------------------|----|-----|
| 31   | <b>DEBUGEV</b> : 指示调试事件发生时间                                        | 读写 | 0x0 |
| 30   | <b>FORCED</b> : 表示具有可配置优先级的故障已升级为 HardFault 异常，原因是该故障因优先级或被禁用而无法激活 | 读写 | 0x0 |
| 29:2 | 保留。                                                                | -  | -   |
| 1    | <b>VECTTBL</b> : 表示由于异常处理过程中向量表读取错误而发生故障                           | 读写 | 0x0 |
| 0    | 保留。                                                                | -  | -   |

## M33: DFSR 寄存器

偏移: 0x0ed30

### 描述

显示发生的调试事件类型

表212. DFSR  
寄存器

| 位    | 描述                                      | 类型 | 复位  |
|------|-----------------------------------------|----|-----|
| 31:5 | 保留。                                     | -  | -   |
| 4    | <b>EXTERNAL</b> : 粘滞标志，指示是否发生外部调试请求调试事件 | 读写 | 0x0 |
| 3    | <b>VCATCH</b> : 粘滞标志，指示是否发生向量捕获调试事件     | 读写 | 0x0 |
| 2    | <b>DWTTRAP</b> : 粘滞标志，指示是否发生观察点调试事件     | 读写 | 0x0 |
| 1    | <b>BKPT</b> : 粘滞标志，指示是否发生断点调试事件         | 读写 | 0x0 |

| 位 | 描述                                         | 类型 | 复位  |
|---|--------------------------------------------|----|-----|
| 0 | <b>HALTED</b> : 粘滞标志，指示是否发生挂起请求调试事件或单步调试事件 | 读写 | 0x0 |

## M33: MMFAR 寄存器

偏移：0x0ed34

### 描述

显示导致 MPU 故障的内存位置地址

表213. MMFAR 寄存器

| 位    | 描述                                                                                                 | 类型 | 复位         |
|------|----------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 地址：该寄存器会更新为引发 MemManage 错误的位置地址。MMFSR 显示错误原因及该字段的有效性。仅当 MMFSR.MMARVALID 被设置时，此字段有效，否则为未知（UNKNOWN）。 | 读写 | 0x00000000 |

## M33: BFAR 寄存器

偏移量：0x0ed38

### 说明

显示与精确数据访问总线错误（BusFault）相关联的地址。

表214. BFAR 寄存器

| 位    | 描述                                                                                    | 类型 | 复位         |
|------|---------------------------------------------------------------------------------------|----|------------|
| 31:0 | 地址：该寄存器会更新为引发 BusFault 的位置地址。BFSR 显示错误原因。仅当 BFSR.BFARVALID 被设置时，此字段有效，否则为未知（UNKNOWN）。 | 读写 | 0x00000000 |

## M33: ID\_PFR0 寄存器

偏移量：0x0ed40

### 说明

提供关于处理引擎（PE）支持的指令集的顶层信息。

表215. ID\_PFR0 寄存器

| 位    | 描述                       | 类型 | 复位  |
|------|--------------------------|----|-----|
| 31:8 | 保留。                      | -  | -   |
| 7:4  | <b>STATE1</b> : 支持T32指令集 | 只读 | 0x3 |
| 3:0  | <b>STATE0</b> : 支持A32指令集 | 只读 | 0x0 |

## M33: ID\_PFR1 寄存器

偏移：0x0ed44

### 描述

提供关于程序员模型及扩展支持的信息

表216. ID\_PFR1 寄存器

| 位     | 描述                                     | 类型 | 复位  |
|-------|----------------------------------------|----|-----|
| 31:12 | 保留。                                    | -  | -   |
| 11:8  | <b>MPROGMOD</b> : 标识对M-Profile程序员模型的支持 | 只读 | 0x5 |
| 7:4   | <b>SECURITY</b> : 标识是否实现安全扩展           | 只读 | 0x2 |
| 3:0   | 保留。                                    | -  | -   |

## M33: ID\_DFR0寄存器

偏移: 0x0ed48

### 描述

提供关于调试系统的顶层信息

表217. ID\_DFR0  
寄存器

| 位     | 描述                                    | 类型 | 复位  |
|-------|---------------------------------------|----|-----|
| 31:24 | 保留。                                   | -  | -   |
| 23:20 | <b>MPROFDBG:</b> 指示所支持的M-Profile 调试架构 | 只读 | 0x2 |
| 19:0  | 保留。                                   | -  | -   |

## M33: ID\_AFR0寄存器

偏移: 0x0ed4c

### 描述

提供PE实现定义特性的信息

表218. ID\_AFR0  
寄存器

| 位     | 描述                     | 类型 | 复位  |
|-------|------------------------|----|-----|
| 31:16 | 保留。                    | -  | -   |
| 15:12 | <b>IMPDEF3:</b> 实现定义含义 | 只读 | 0x0 |
| 11:8  | <b>IMPDEF2:</b> 实现定义含义 | 只读 | 0x0 |
| 7:4   | <b>IMPDEF1:</b> 实现定义含义 | 只读 | 0x0 |
| 3:0   | <b>IMPDEF0:</b> 实现定义含义 | 只读 | 0x0 |

## M33: ID\_MMFR0 寄存器

偏移: 0x0ed50

### 描述

提供已实现的内存模型及内存管理支持的相关信息

表219. ID\_MMFR0寄  
存器

| 位     | 描述                                    | 类型 | 复位  |
|-------|---------------------------------------|----|-----|
| 31:24 | 保留。                                   | -  | -   |
| 23:20 | <b>AUXREG:</b> 指示辅助控制寄存器的支持情况         | 只读 | 0x1 |
| 19:16 | <b>TCM:</b> 指示紧耦合存储器 (TCMs) 的支持情况     | 只读 | 0x0 |
| 15:12 | <b>SHARELVL:</b> 指示所实现的共享级别数量         | 只读 | 0x1 |
| 11:8  | <b>OUTERSHR:</b> 指示所实现的最外层共享域         | 只读 | 0xf |
| 7:4   | <b>PMSA:</b> 指示受保护内存系统架构 (PMSA) 的支持情况 | 只读 | 0x4 |
| 3:0   | 保留。                                   | -  | -   |

## M33: ID\_MMFR1 寄存器

偏移: 0x0ed54

**描述**

提供已实现的内存模型及内存管理支持的相关信息

表220. ID\_MMFR1寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33：ID\_MMFR2 寄存器**

偏移：0x0ed58

**描述**

提供已实现的内存模型及内存管理支持的相关信息

表221. ID\_MMFR2寄存器

| 位     | 描述                                    | 类型 | 复位  |
|-------|---------------------------------------|----|-----|
| 31:28 | 保留。                                   | -  | -   |
| 27:24 | <b>WFISTALL</b> ：指示等待中断（WFI）暂停功能的支持情况 | 只读 | 0x1 |
| 23:0  | 保留。                                   | -  | -   |

**M33：ID\_MMFR3 寄存器**

偏移量：0x0ed5c

**描述**

提供已实现的内存模型及内存管理支持的相关信息

表 222. ID\_MMFR3 寄存器

| 位     | 描述                               | 类型 | 复位  |
|-------|----------------------------------|----|-----|
| 31:12 | 保留。                              | -  | -   |
| 11:8  | <b>BPMINT</b> ：指示支持的分支预测器维护      | 只读 | 0x0 |
| 7:4   | <b>CMAINTSW</b> ：指示按组/路支持的缓存维护操作 | 只读 | 0x0 |
| 3:0   | <b>CMAINTVA</b> ：指示按地址支持的缓存维护操作  | 只读 | 0x0 |

**M33：ID\_ISAR0 寄存器**

偏移量：0x0ed60

**描述**

提供有关 PE 实现的指令集的信息

表 223. ID\_ISAR0 寄存器

| 位     | 描述                               | 类型 | 复位  |
|-------|----------------------------------|----|-----|
| 31:28 | 保留。                              | -  | -   |
| 27:24 | <b>DIVIDE</b> ：指示支持的除法指令         | 只读 | 0x8 |
| 23:20 | <b>DEBUG</b> ：指示已实现的调试指令         | 只读 | 0x0 |
| 19:16 | <b>COPROC</b> ：指示支持的协处理器指令       | 只读 | 0x9 |
| 15:12 | <b>CMPBRANCH</b> ：指示支持的组合比较与分支指令 | 只读 | 0x2 |
| 11:8  | <b>BITFIELD</b> ：指示支持的位字段指令      | 只读 | 0x3 |

| 位   | 描述                           | 类型 | 复位  |
|-----|------------------------------|----|-----|
| 7:4 | <b>BITCOUNT</b> : 指示支持的位计数指令 | 只读 | 0x0 |
| 3:0 | 保留。                          | -  | -   |

## M33: ID\_ISAR1 寄存器

偏移量: 0x0ed64

### 描述

提供有关 PE 实现的指令集的信息

表224. ID\_ISAR1  
寄存器

| 位     | 描述                                   | 类型 | 复位  |
|-------|--------------------------------------|----|-----|
| 31:28 | 保留。                                  | -  | -   |
| 27:24 | <b>INTERWORK</b> : 表示已实现的交互指令        | 只读 | 0x5 |
| 23:20 | <b>IMMEDIATE</b> : 表示已实现的带长立即数数据处理指令 | 只读 | 0x7 |
| 19:16 | <b>IFTHEN</b> : 表示已实现的条件执行指令         | 只读 | 0x2 |
| 15:12 | <b>EXTEND</b> : 表示已实现的扩展指令           | 只读 | 0x5 |
| 11:0  | 保留。                                  | -  | -   |

## M33: ID\_ISAR2 寄存器

偏移: 0x0ed68

### 描述

提供有关 PE 实现的指令集的信息

表225. ID\_ISAR2  
寄存器

| 位     | 描述                                    | 类型 | 复位  |
|-------|---------------------------------------|----|-----|
| 31:28 | <b>REVERSAL</b> : 表示已实现的反转指令          | 只读 | 0x3 |
| 27:24 | 保留。                                   | -  | -   |
| 23:20 | <b>MULTU</b> : 表示已实现的高级无符号乘法指令        | 只读 | 0x1 |
| 19:16 | <b>MULTS</b> : 表示已实现的高级有符号乘法指令        | 只读 | 0x7 |
| 15:12 | <b>MULT</b> : 表示已实现的额外乘法指令            | 只读 | 0x3 |
| 11:8  | <b>MULTIACCESSINT</b> : 表示支持可中断多重访问指令 | 只读 | 0x4 |
| 7:4   | <b>MEMHINT</b> : 表示已实现的内存提示指令         | 只读 | 0x2 |
| 3:0   | <b>LOADSTORE</b> : 表示已实现的额外加载/存储指令    | 只读 | 0x6 |

## M33: ID\_ISAR3 寄存器

偏移: 0x0ed6c

### 描述

提供有关 PE 实现的指令集的信息

表226. ID\_ISAR3  
寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:28 | 保留。 | -  | -  |

| 位     | 描述                                                           | 类型 | 复位  |
|-------|--------------------------------------------------------------|----|-----|
| 27:24 | <b>TRUE NOP</b> : 表示已实现的真实NOP指令                              | 只读 | 0x7 |
| 23:20 | <b>T32COPY</b> : 表示支持T32非标志位设置的MOV指令                         | 只读 | 0x8 |
| 19:16 | <b>TABBRANCH</b> : 表示已实现的表分支指令                               | 只读 | 0x9 |
| 15:12 | <b>SYNCHPRIM</b> : 与ID_ISAR4.SynchPrim_frac配合使用，表示已实现的同步原语指令 | 只读 | 0x5 |
| 11:8  | <b>SVC</b> : 表示已实现的SVC指令                                     | 只读 | 0x7 |
| 7:4   | <b>SIMD</b> : 表示已实现的SIMD指令                                   | 只读 | 0x2 |
| 3:0   | <b>SATURATE</b> : 表示已实现的饱和指令                                 | 只读 | 0x9 |

## M33: ID\_ISAR4 寄存器

偏移: 0x0ed70

### 描述

提供有关 PE 实现的指令集的信息

表227. ID\_ISAR4 寄存器

| 位     | 描述                                                          | 类型 | 复位  |
|-------|-------------------------------------------------------------|----|-----|
| 31:28 | 保留。                                                         | -  | -   |
| 27:24 | <b>PSR_M</b> : 表示已实现用于修改PSRs的M配置文件指令                        | 只读 | 0x1 |
| 23:20 | <b>SYNCPROM_FRAC</b> : 与ID_ISAR3.SynchPrim配合使用，表示已实现的同步原语指令 | 只读 | 0x3 |
| 19:16 | <b>BARRIER</b> : 表示已实现的屏障指令                                 | 只读 | 0x1 |
| 15:12 | 保留。                                                         | -  | -   |
| 11:8  | <b>WRITEBACK</b> : 表示支持写回寻址模式                               | 只读 | 0x1 |
| 7:4   | <b>WITHSHIFTS</b> : 表示对回写寻址模式的支持                            | 只读 | 0x3 |
| 3:0   | <b>UNPRIV</b> : 表示已实现的非特权指令                                 | 只读 | 0x2 |

## M33: ID\_ISAR5 寄存器

偏移量: 0x0ed74

### 描述

提供有关 PE 实现的指令集的信息

表 228. ID\_ISAR5 寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: CTR 寄存器

偏移量: 0x0ed7c

### 描述

提供有关缓存架构的信息当CLIDR为零时，CTR为保留值0 (RES0)

表 229. CTR 寄存器

| 位  | 描述                    | 类型 | 复位  |
|----|-----------------------|----|-----|
| 31 | <b>RES1</b> : 保留，RES1 | 只读 | 0x1 |

| 位     | 描述                                                                            | 类型 | 复位  |
|-------|-------------------------------------------------------------------------------|----|-----|
| 30:28 | 保留。                                                                           | -  | -   |
| 27:24 | <b>CWG</b> : 因驱逐包含已修改内存位置的缓存条目所能覆盖的最大内存大小（以字为单位）的对数（以2为底）                     | 只读 | 0x0 |
| 23:20 | <b>ERG</b> : 为 Load-Exclusive 与 Store-Exclusive 指令实现的最大保留粒度大小（以字为单位）的对数（以2为底） | 只读 | 0x0 |
| 19:16 | <b>DMINLINE</b> : PE 控制的所有数据缓存和统一缓存中最小缓存行大小（以字为单位）的对数（以2为底）                   | 只读 | 0x0 |
| 15:14 | <b>RES1_1</b> : 保留，RES1                                                       | 只读 | 0x3 |
| 13:4  | 保留。                                                                           | -  | -   |
| 3:0   | <b>IMINLINE</b> : PE 控制的所有指令缓存中最小缓存行大小（以字为单位）的对数（以2为底）                        | 只读 | 0x0 |

## M33: CPACR 寄存器

偏移: 0x0ed88

### 描述

指定协处理器及浮点扩展的访问权限

表230. CPACR寄存器

| 位     | 描述                                                                       | 类型 | 复位  |
|-------|--------------------------------------------------------------------------|----|-----|
| 31:24 | 保留。                                                                      | -  | -   |
| 23:22 | <b>CP11</b> : 此字段值将被忽略。若未实现浮点扩展，此字段为 RAZ/WI。若此位的值未设置为与 CP10 字段相同，则该值为未知。 | 读写 | 0x0 |
| 21:20 | <b>CP10</b> : 定义浮点功能的访问权限                                                | 读写 | 0x0 |
| 19:16 | 保留。                                                                      | -  | -   |
| 15:14 | <b>CP7</b> : 控制协处理器7的访问权限                                                | 读写 | 0x0 |
| 13:12 | <b>CP6</b> : 控制协处理器6的访问权限                                                | 读写 | 0x0 |
| 11:10 | <b>CP5</b> : 控制协处理器5的访问权限                                                | 读写 | 0x0 |
| 9:8   | <b>CP4</b> : 控制协处理器4的访问权限                                                | 读写 | 0x0 |
| 7:6   | <b>CP3</b> : 控制协处理器3的访问权限                                                | 读写 | 0x0 |
| 5:4   | <b>CP2</b> : 控制协处理器2的访问权限                                                | 读写 | 0x0 |
| 3:2   | <b>CP1</b> : 控制协处理器1的访问权限                                                | 读写 | 0x0 |
| 1:0   | <b>CP0</b> : 控制协处理器0的访问权限                                                | 读写 | 0x0 |

## M33: NSACR 寄存器

偏移: 0x0ed8c

### 描述

定义浮点扩展及协处理器 CP0 至 CP7 的非安全访问权限

表231. NSACR 寄存器

| 位     | 描述                         | 类型 | 复位  |
|-------|----------------------------|----|-----|
| 31:12 | 保留。                        | -  | -   |
| 11    | <b>CP11:</b> 启用非安全访问浮点扩展   | 读写 | 0x0 |
| 10    | <b>CP10:</b> 启用非安全访问浮点扩展   | 读写 | 0x0 |
| 9:8   | 保留。                        | -  | -   |
| 7     | <b>CP7:</b> 启用非安全访问协处理器CP7 | 读写 | 0x0 |
| 6     | <b>CP6:</b> 启用非安全访问协处理器CP6 | 读写 | 0x0 |
| 5     | <b>CP5:</b> 启用非安全访问协处理器CP5 | 读写 | 0x0 |
| 4     | <b>CP4:</b> 启用非安全访问协处理器CP4 | 读写 | 0x0 |
| 3     | <b>CP3:</b> 启用非安全访问协处理器CP3 | 读写 | 0x0 |
| 2     | <b>CP2:</b> 启用非安全访问协处理器CP2 | 读写 | 0x0 |
| 1     | <b>CP1:</b> 启用非安全访问协处理器CP1 | 读写 | 0x0 |
| 0     | <b>CP0:</b> 启用非安全访问协处理器CP0 | 读写 | 0x0 |

## M33: MPU\_TYPE 寄存器

偏移: 0x0ed90

### 描述

MPU 类型寄存器指示 MPU `FTSSS 支持的区域数量

表232. MPU\_TYPE 寄存器

| 位     | 描述                                | 类型 | 复位   |
|-------|-----------------------------------|----|------|
| 31:16 | 保留。                               | -  | -    |
| 15:8  | <b>DREGION:</b> MPU 支持的区域数量       | 只读 | 0x08 |
| 7:1   | 保留。                               | -  | -    |
| 0     | <b>SEPARATE:</b> 表示支持独立的指令和数据地址区域 | 只读 | 0x0  |

## M33: MPU\_CTRL 寄存器

偏移: 0x0ed94

### 描述

使能 MPU；当 MPU 使能时，控制是否将默认内存映射作为特权访问的背景区域，以及在 FAULTMASK 置为 1 时，是否对 HardFault、NMI 及异常处理程序启用 MPU

表233. MPU\_CTRL 寄存器

| 位    | 描述                                                                                                  | 类型 | 复位  |
|------|-----------------------------------------------------------------------------------------------------|----|-----|
| 31:3 | 保留。                                                                                                 | -  | -   |
| 2    | <b>PRIVDEFENA:</b> 控制是否为特权软件启用默认内存映射                                                                | 读写 | 0x0 |
| 1    | <b>HFNMIENA:</b> 控制优先级低于 0 的处理程序在 MPU 启用或禁用时访问内存<br>此项适用于 HardFault、NMI 以及 FAULTMASK 设置为 1 时的异常处理程序 | 读写 | 0x0 |
| 0    | <b>ENABLE:</b> 启用 MPU                                                                               | 读写 | 0x0 |

## M33: MPU\_RNR 寄存器

偏移: 0x0ed98

### 描述

选择当前由 MPU\_RBAR 和 MPU\_RLAR 访问的区域

表234. MPU\_RNR寄存器

| 位    | 描述                                            | 类型 | 复位  |
|------|-----------------------------------------------|----|-----|
| 31:3 | 保留。                                           | -  | -   |
| 2:0  | <b>REGION:</b> 指示 MPU_RBAR 与 MPU_RLAR 访问的内存区域 | 读写 | 0x0 |

## M33: MPU\_RBAR 寄存器

偏移: 0x0ed9c

### 描述

提供对当前选定 MPU 区域 `FTSSS 基地址的间接读写访问

表235. MPU\_RBAR寄存器

| 位    | 描述                                                            | 类型 | 复位         |
|------|---------------------------------------------------------------|----|------------|
| 31:5 | <b>BASE:</b> 包含所选 MPU 内存区域下限的[31:5]位（含端点）。该值通过零扩展以提供需匹配检测的基址。 | 读写 | 0x00000000 |
| 4:3  | <b>SH:</b> 定义该区域在普通内存中的共享域属性。                                 | 读写 | 0x0        |
| 2:1  | <b>AP:</b> 定义该区域的访问权限。                                        | 读写 | 0x0        |
| 0    | <b>XN:</b> 定义是否允许从该区域执行代码。                                    | 读写 | 0x0        |

## M33: MPU\_RLAR 寄存器

偏移: 0x0eda0

### 描述

提供对当前选定 MPU 区域 `FTSSS 限制地址的间接读写访问

表236. MPU\_RLAR寄存器

| 位    | 描述                                                                    | 类型 | 复位         |
|------|-----------------------------------------------------------------------|----|------------|
| 31:5 | <b>LIMIT:</b> 包含所选 MPU 内存区域上限的[31:5]位（含端点）。该值后缀附加 0x1F，以提供需匹配检测的限止地址。 | 读写 | 0x00000000 |
| 4    | 保留。                                                                   | -  | -          |
| 3:1  | <b>ATTRINDEX:</b> 关联 MPU_MAIR0 和 MPU_MAIR1 字段中的一组属性。                  | 读写 | 0x0        |
| 0    | <b>EN:</b> 区域使能                                                       | 读写 | 0x0        |

## M33: MPU\_RBAR\_A1 寄存器

偏移: 0x0eda4

### 描述

提供对由 MPU\_RNR[7:2] 选择的 MPU 区域地址的间接读写访问：(1[1:0])`FTSSS

表 237。  
MPU\_RBAR\_A1  
寄存器

| 位    | 描述                                                              | 类型 | 复位        |
|------|-----------------------------------------------------------------|----|-----------|
| 31:5 | <b>BASE</b> : 包含所选 MPU 内存区域下限的[31:5]位（含端点）。该值通过零扩展以提供需匹配检测的基地址。 | 读写 | 0x0000000 |
| 4:3  | <b>SH</b> : 定义该区域在普通内存中的共享域属性。                                  | 读写 | 0x0       |
| 2:1  | <b>AP</b> : 定义该区域的访问权限。                                         | 读写 | 0x0       |
| 0    | <b>XN</b> : 定义是否允许从该区域执行代码。                                     | 读写 | 0x0       |

## M33: MPU\_RLAR\_A1 寄存器

偏移: 0x0eda8

### 描述

提供对由 MPU\_RNR[7:2](1[1:0]) 选择的当前 MPU 区域限制地址的间接读写访问 `FTSSS

表 238。  
MPU\_RLAR\_A1  
寄存器

| 位    | 描述                                                                     | 类型 | 复位        |
|------|------------------------------------------------------------------------|----|-----------|
| 31:5 | <b>LIMIT</b> : 包含所选 MPU 内存区域上限的[31:5]位（含端点）。该值后缀附加 0x1F，以提供需匹配检测的限止地址。 | 读写 | 0x0000000 |
| 4    | 保留。                                                                    | -  | -         |
| 3:1  | <b>ATTRINDEX</b> : 关联 MPU_MAIRO 和 MPU_MAIR1 字段中的一组属性。                  | 读写 | 0x0       |
| 0    | <b>EN</b> : 区域使能                                                       | 读写 | 0x0       |

## M33: MPU\_RBAR\_A2 寄存器

偏移: 0x0edac

### 描述

提供对由 MPU\_RNR[7:2] 选择的 MPU 区域地址的间接读写访问：(2[1:0])  
`FTSSS

表 239。  
MPU\_RBAR\_A2  
寄存器

| 位    | 描述                                                              | 类型 | 复位        |
|------|-----------------------------------------------------------------|----|-----------|
| 31:5 | <b>BASE</b> : 包含所选 MPU 内存区域下限的[31:5]位（含端点）。该值通过零扩展以提供需匹配检测的基地址。 | 读写 | 0x0000000 |
| 4:3  | <b>SH</b> : 定义该区域在普通内存中的共享域属性。                                  | 读写 | 0x0       |
| 2:1  | <b>AP</b> : 定义该区域的访问权限。                                         | 读写 | 0x0       |
| 0    | <b>XN</b> : 定义是否允许从该区域执行代码。                                     | 读写 | 0x0       |

## M33: MPU\_RLAR\_A2 寄存器

偏移: 0x0edb0

### 描述

提供对由 MPU\_RNR[7:2](2[1:0]) 选择的当前 MPU 区域限制地址的间接读写访问 `FTSSS

表240。  
MPU\_RLAR\_A2  
寄存器

| 位    | 描述                                                                     | 类型 | 复位        |
|------|------------------------------------------------------------------------|----|-----------|
| 31:5 | <b>LIMIT</b> : 包含所选 MPU 内存区域上限的[31:5]位（含端点）。该值后缀附加 0x1F，以提供需匹配检测的限止地址。 | 读写 | 0x0000000 |
| 4    | 保留。                                                                    | -  | -         |
| 3:1  | <b>ATTRINDEX</b> : 关联 MPU_MAIR0 和 MPU_MAIR1 字段中的一组属性。                  | 读写 | 0x0       |
| 0    | <b>EN</b> : 区域使能                                                       | 读写 | 0x0       |

## M33: MPU\_RBAR\_A3寄存器

偏移: 0x0edb4

### 描述

提供对由MPU\_RNR[7:2]:3[1:0])选定的MPU区域基址的间接读写访问  
'FTSSS'

表241。  
MPU\_RBAR\_A3  
寄存器

| 位    | 描述                                                              | 类型 | 复位        |
|------|-----------------------------------------------------------------|----|-----------|
| 31:5 | <b>BASE</b> : 包含所选 MPU 内存区域下限的[31:5]位（含端点）。该值通过零扩展以提供需匹配检测的基地址。 | 读写 | 0x0000000 |
| 4:3  | <b>SH</b> : 定义该区域在普通内存中的共享域属性。                                  | 读写 | 0x0       |
| 2:1  | <b>AP</b> : 定义该区域的访问权限。                                         | 读写 | 0x0       |
| 0    | <b>XN</b> : 定义是否允许从该区域执行代码。                                     | 读写 | 0x0       |

## M33: MPU\_RLAR\_A3寄存器

偏移: 0x0edb8

### 描述

提供对当前由 MPU\_RNR[7:2]:3[1:0]) 'FTSSS' 选择的 MPU 区域限制地址的间接读写访问

表242。  
MPU\_RLAR\_A3  
寄存器

| 位    | 描述                                                                     | 类型 | 复位        |
|------|------------------------------------------------------------------------|----|-----------|
| 31:5 | <b>LIMIT</b> : 包含所选 MPU 内存区域上限的[31:5]位（含端点）。该值后缀附加 0x1F，以提供需匹配检测的限止地址。 | 读写 | 0x0000000 |
| 4    | 保留。                                                                    | -  | -         |
| 3:1  | <b>ATTRINDEX</b> : 关联 MPU_MAIR0 和 MPU_MAIR1 字段中的一组属性。                  | 读写 | 0x0       |
| 0    | <b>EN</b> : 区域使能                                                       | 读写 | 0x0       |

## M33: MPU\_MAIR0寄存器

偏移: 0x0edc0

### 描述

与 MPU\_MAIR1 一起，提供对应于 AttrIndex 值的内存属性编码

表243。  
MPU\_MAIR0寄存器

| 位     | 描述                                      | 类型 | 复位   |
|-------|-----------------------------------------|----|------|
| 31:24 | <b>ATTR3</b> : AttrIndex为3的MPU区域的内存属性编码 | 读写 | 0x00 |

| 位     | 描述                                        | 类型 | 复位   |
|-------|-------------------------------------------|----|------|
| 23:16 | <b>ATTR2</b> : AttrIndex为2的MPU区域的内存属性编码   | 读写 | 0x00 |
| 15:8  | <b>ATTR1</b> : AttrIndex为1的MPU区域的内存属性编码   | 读写 | 0x00 |
| 7:0   | <b>ATTR0</b> : 用于AttrIndex为0的MPU区域的内存属性编码 | 读写 | 0x00 |

## M33: MPU\_MAIR1 寄存器

偏移: 0x0edc4

### 描述

与 MPU\_MAIRO 一起, 提供对应于 AttrIndex 值的内存属性编码

表244。  
MPU\_MAIR1 寄存器

| 位     | 描述                                        | 类型 | 复位   |
|-------|-------------------------------------------|----|------|
| 31:24 | <b>ATTR7</b> : 用于AttrIndex为7的MPU区域的内存属性编码 | 读写 | 0x00 |
| 23:16 | <b>ATTR6</b> : 用于AttrIndex为6的MPU区域的内存属性编码 | 读写 | 0x00 |
| 15:8  | <b>ATTR5</b> : 用于AttrIndex为5的MPU区域的内存属性编码 | 读写 | 0x00 |
| 7:0   | <b>ATTR4</b> : 用于AttrIndex为4的MPU区域的内存属性编码 | 读写 | 0x00 |

## M33: SAU\_CTRL 寄存器

偏移: 0x0edd0

### 描述

允许启用安全属性单元

表245. SAU\_CTRL寄存器

| 位    | 描述                                                   | 类型 | 复位  |
|------|------------------------------------------------------|----|-----|
| 31:2 | 保留。                                                  | -  | -   |
| 1    | <b>ALLNS</b> : 当SAU_CTRL.ENABLE为0时, 该位控制内存的标记为非安全或安全 | 读写 | 0x0 |
| 0    | <b>ENABLE</b> : 启用SAU                                | 读写 | 0x0 |

## M33: SAU\_TYPE 寄存器

偏移: 0x0edd4

### 描述

指示安全归属单元 (SAU) 已实现的区域数

表246. SAU\_TYPE  
寄存器

| 位    | 描述                          | 类型 | 复位   |
|------|-----------------------------|----|------|
| 31:8 | 保留。                         | -  | -    |
| 7:0  | <b>SREGION</b> : 已实现的SAU区域数 | 只读 | 0x08 |

## M33: SAU\_RNR寄存器

偏移: 0x0edd8

### 描述

选择当前由 SAU\_RBAR 和 SAU\_RLAR 访问的区域

表247. SAU\_RNR  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:8 | 保留。 | -  | -  |

| 位   | 描述                                         | 类型 | 复位   |
|-----|--------------------------------------------|----|------|
| 7:0 | <b>REGION:</b> 指示SAU_RBAR与SAU_RLAR访问的SAU区域 | 读写 | 0x00 |

## M33: SAU\_RBAR寄存器

偏移: 0x0eddc

### 描述

提供对当前选定 SAU 区域基址的间接读写访问

表248. SAU\_RBAR  
寄存器

| 位    | 描述                                 | 类型 | 复位         |
|------|------------------------------------|----|------------|
| 31:5 | <b>BADDR:</b> 存储所选SAU区域基地址的位[31:5] | 读写 | 0x00000000 |
| 4:0  | 保留。                                | -  | -          |

## M33: SAU\_RLAR寄存器

偏移: 0x0ede0

### 描述

提供对当前选定 SAU 区域限制地址的间接读写访问

表249. SAU\_RLAR  
寄存器

| 位    | 描述                                  | 类型 | 复位         |
|------|-------------------------------------|----|------------|
| 31:5 | <b>LADDR:</b> 存储所选SAU区域限制地址的位[31:5] | 读写 | 0x00000000 |
| 4:2  | 保留。                                 | -  | -          |
| 1    | <b>NSC:</b> 控制非安全状态是否允许从该区域执行SG指令   | 读写 | 0x0        |
| 0    | <b>ENABLE:</b> SAU区域使能              | 读写 | 0x0        |

## M33: SFSR寄存器

偏移: 0x0ede4

### 描述

提供有关任一安全相关故障的信息

表250. SFSR  
寄存器

| 位    | 描述                                                                                                   | 类型 | 复位  |
|------|------------------------------------------------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                                                                  | -  | -   |
| 7    | <b>LSERR:</b> 粘滞标志，指示在延迟状态激活或停用期间发生错误                                                                | 读写 | 0x0 |
| 6    | <b>SFARVALID:</b> 当SFAR寄存器包含有效值时，此位被置位。与类似字段如BFSR.BFARVALID和MMFSR.MMARVALID相同，此位可被其他异常（如BusFault）清除。 | 读写 | 0x0 |
| 5    | <b>LSPERR:</b> 粘滞标志，指示在延迟保存浮点状态期间发生了SAU或IDAU违规。                                                      | 读写 | 0x0 |
| 4    | <b>INVTRAN:</b> 粘滞标志，指示由于分支未标记为跨域，导致从安全到非安全内存转换时引发异常。                                                | 读写 | 0x0 |

| 位 | 描述                                                                                                | 类型 | 复位  |
|---|---------------------------------------------------------------------------------------------------|----|-----|
| 3 | <b>AUVOIOL:</b> 粘滞标志，指示尝试访问被标记为安全的地址空间且事务集设置为非安全的NS-Req部分。若违规发生在延迟状态保存期间，此位不被置位。<br>详见LSPERR。     | 读写 | 0x0 |
| 2 | <b>INVER:</b> 该情况可能由从非安全状态异常返回时，EXC_RETURN.DCRS被设置为0，或者EXC_RETURN.ES被设置为1导致。                      | 读写 | 0x0 |
| 1 | <b>INVIS:</b> 如果在取消堆栈操作过程中发现异常堆栈帧中的完整性签名无效，则设置该位。                                                 | 读写 | 0x0 |
| 0 | <b>INVEP:</b> 如果来自非安全状态的函数调用或异常定位于安全状态中的非SG指令，则设置该位。如果目标地址为SG指令，但没有匹配且NSC标志被设置的SAU/IDAU区域，也会设置该位。 | 读写 | 0x0 |

## M33: SFAR 寄存器

偏移量: 0x0ede8

### 说明

显示导致安全违规的内存位置地址。

表 251. SFAR 寄存器

| 位    | 描述                                                                                                                                                                                 | 类型 | 复位         |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 地址: 导致属性单元违规的访问地址。<br>仅当 SFSR.SFARVALID 被设置时，此字段方有效。该设置允许与此寄存器相关的实际触发器与其他故障地址寄存器共享。若实现选择以此方式共享存储，必须谨慎防止将安全地址信息泄露至非安全状态。<br><br>实现该目的的一种方法是将 SFAR 寄存器与 MMFAR_S 寄存器共享，而后者对非安全状态不可访问。 | 读写 | 0x00000000 |

## M33: DHCSR 寄存器

偏移: 0x0edf0

### 说明

控制调试中断

表 252. DHCSR 寄存器

| 位     | 描述                                                                                               | 类型 | 复位  |
|-------|--------------------------------------------------------------------------------------------------|----|-----|
| 31:27 | 保留。                                                                                              | -  | -   |
| 26    | <b>S_RESTART_ST:</b> 指示处理器已处理将 DHCSR.C_HALT 清零（设为0）的请求。即写入 DHCSR 以将 DHCSR.C_HALT 从1清零为0，或外部重启请求。 | 只读 | 0x0 |
| 25    | <b>S_RESET_ST:</b> 指示自上次读取 DHCSR 以来处理器是否已复位。                                                     | 只读 | 0x0 |
| 24    | <b>S_RETIRE_ST:</b> 每当处理器退役一条或多条指令时，此位被置为1。                                                      | 只读 | 0x0 |
| 23:21 | 保留。                                                                                              | -  | -   |
| 20    | <b>S_SDE:</b> 指示是否允许安全侵入式调试                                                                      | 只读 | 0x0 |
| 19    | <b>S_LOCKUP:</b> 指示处理单元（PE）是否处于锁定状态                                                              | 只读 | 0x0 |
| 18    | <b>S_SLEEP:</b> 指示处理单元（PE）是否处于睡眠状态                                                               | 只读 | 0x0 |

| 位    | 描述                                                        | 类型 | 复位  |
|------|-----------------------------------------------------------|----|-----|
| 17   | <b>S_HALT:</b> 指示处理单元（PE）是否处于调试状态                         | 只读 | 0x0 |
| 16   | <b>S_REGRDY:</b> 通过DCRDR传输的握手标志                           | 只读 | 0x0 |
| 15:6 | 保留。                                                       | -  | -   |
| 5    | <b>C_SNAPSTALL:</b> 允许不精确地进入调试状态                          | 读写 | 0x0 |
| 4    | 保留。                                                       | -  | -   |
| 3    | <b>C_MASKINTS:</b> 调试启用时，调试器可写此位以屏蔽PendSV、SysTick及外部可配置中断 | 读写 | 0x0 |
| 2    | <b>C_STEP:</b> 启用单指令步进                                    | 读写 | 0x0 |
| 1    | <b>C_HALT:</b> 处理单元进入调试状态的停止请求                            | 读写 | 0x0 |
| 0    | <b>C_DEBUGEN:</b> 启用停机调试                                  | 读写 | 0x0 |

## M33: DCRSR寄存器

偏移: 0x0edf4

### 描述

与 DCRSR 协同，提供对通用寄存器、特定用途寄存器及浮点扩展寄存器的调试访问。写入 DCRSR 以指定传输的寄存器、传输类型（读或写），并启动传输

表253. DCRSR 寄存器

| 位     | 描述                                    | 类型 | 复位   |
|-------|---------------------------------------|----|------|
| 31:17 | 保留。                                   | -  | -    |
| 16    | <b>REGWR:</b> 指定传输的访问类型               | 读写 | 0x0  |
| 15:7  | 保留。                                   | -  | -    |
| 6:0   | <b>REGSEL:</b> 指定传输的通用寄存器、特殊寄存器或浮点寄存器 | 读写 | 0x00 |

## M33: DCRDR寄存器

偏移: 0x0edf8

### 描述

与 DCRSR 协同，提供对通用寄存器、特定用途寄存器及浮点扩展寄存器的调试访问。若实现主扩展，还可用于外部调试器与运行于处理单元（PE）上的调试代理之间的消息传递

表254. DCRDR 寄存器

| 位    | 描述                                               | 类型 | 复位         |
|------|--------------------------------------------------|----|------------|
| 31:0 | <b>DBGTMP:</b> 提供调试访问，用于读取和写入通用寄存器、特殊寄存器及浮点扩展寄存器 | 读写 | 0x00000000 |

## M33: DEMCR寄存器

偏移: 0x0edfc

### 描述

管理向量捕获行为及调试监视器在调试过程中的处理

表255. DEMCR 寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:25 | 保留。 | -  | -  |

| 位     | 描述                                                              | 类型 | 复位  |
|-------|-----------------------------------------------------------------|----|-----|
| 24    | <b>TRCENA</b> : 启用所有DWT和ITM功能的全局开关                              | 读写 | 0x0 |
| 23:21 | 保留。                                                             | -  | -   |
| 20    | <b>SDME</b> : 指示调试监视器是否针对安全态或非安全态, 以及是否允许在安全态发生调试事件             | 只读 | 0x0 |
| 19    | <b>MON_REQ</b> : 调试监视器信号量位                                      | 读写 | 0x0 |
| 18    | <b>MON_STEP</b> : 启用调试监视器单步执行                                   | 读写 | 0x0 |
| 17    | <b>MON_PEND</b> : 设置或清除调试监视器异常的挂起状态                             | 读写 | 0x0 |
| 16    | <b>MON_EN</b> : 启用调试监视器异常                                       | 读写 | 0x0 |
| 15:12 | 保留。                                                             | -  | -   |
| 11    | <b>VC_SFERR</b> : 安全错误异常中止调试向量捕获的启用                             | 读写 | 0x0 |
| 10    | <b>VC_HARDERR</b> : 硬错误异常中止调试向量捕获的启用                            | 读写 | 0x0 |
| 9     | <b>VC_INTEERR</b> : 启用在异常入口和返回期间因故障而中止调试向量捕获                    | 读写 | 0x0 |
| 8     | <b>VC_BUSERR</b> : 启用总线错误异常的中止调试向量捕获                            | 读写 | 0x0 |
| 7     | <b>VC_STATERR</b> : 启用因状态信息错误 (例如未定义指令异常) 导致的UsageFault异常中止调试陷阱 | 读写 | 0x0 |
| 6     | <b>VC_CHKERR</b> : 启用因校验错误 (例如对齐校验错误) 导致的UsageFault异常中止调试陷阱     | 读写 | 0x0 |
| 5     | <b>VC_NOCPERR</b> : 启用因访问协处理器导致的UsageFault异常中止调试陷阱              | 读写 | 0x0 |
| 4     | <b>VC_MMERR</b> : 启用发生内存管理 (MemManage) 异常时的中止调试陷阱               | 读写 | 0x0 |
| 3:1   | 保留。                                                             | -  | -   |
| 0     | <b>VC_CORERESET</b> : 启用复位向量捕获。此功能会使暖复位中止正在运行的系统                | 读写 | 0x0 |

## M33: DCSR寄存器

偏移: 0x0ee08

### 描述

提供安全调试的控制与状态信息

表256. DCSR  
寄存器

| 位     | 描述                                                              | 类型 | 复位  |
|-------|-----------------------------------------------------------------|----|-----|
| 31:18 | 保留。                                                             | -  | -   |
| 17    | <b>CDSKEY</b> : 除非同时将CDSKEY写入零, 否则对CDS位的写入将被忽略                  | 读写 | 0x0 |
| 16    | <b>CDS</b> : 本字段指示处理器当前的安全状态                                    | 读写 | 0x0 |
| 15:2  | 保留。                                                             | -  | -   |
| 1     | <b>SBRSEL</b> : 当 SBRSELEN 为 1 时, 该位选择调试器访问非安全版本或安全版本的内存映射银行寄存器 | 读写 | 0x0 |

| 位 | 描述                                                            | 类型 | 复位  |
|---|---------------------------------------------------------------|----|-----|
| 0 | <b>SBRSELEN:</b> 控制由 SBRSEL 字段还是处理器当前的安全状态决定调试器访问的内存映射银行寄存器版本 | 读写 | 0x0 |

## M33: STIR 寄存器

偏移: 0x0ef00

### 描述

为软件生成中断提供机制

表 257. STIR 寄存器

| 位    | 描述                                                | 类型 | 复位    |
|------|---------------------------------------------------|----|-------|
| 31:9 | 保留。                                               | -  | -     |
| 8:0  | <b>INTID:</b> 指示要挂起的中断写入值为 (ExceptionNumber - 16) | 读写 | 0x000 |

## M33: FPCCR 寄存器

偏移: 0x0ef34

### 描述

存储浮点扩展的控制数据

表 258. FPCCR 寄存器

| 位     | 描述                                                                                      | 类型 | 复位  |
|-------|-----------------------------------------------------------------------------------------|----|-----|
| 31    | <b>ASPEN:</b> 该位置1时，执行浮点指令会将 CONTROL.FPCA 位置1                                           | 读写 | 0x0 |
| 30    | <b>LSPEN:</b> 启用浮点状态的延迟上下文保存                                                            | 读写 | 0x0 |
| 29    | <b>LSPENS:</b> 此位控制LSPEN位是否允许从非安全态写入                                                    | 读写 | 0x1 |
| 28    | <b>CLRONRET:</b> 在异常返回时清除浮点调用者保存寄存器                                                     | 读写 | 0x0 |
| 27    | <b>CLRONRETS:</b> 此位控制CLRONRET位是否允许从非安全态写入                                              | 读写 | 0x0 |
| 26    | <b>TS:</b> 将浮点寄存器视为启用安全状态                                                               | 读写 | 0x0 |
| 25:11 | 保留。                                                                                     | -  | -   |
| 10    | <b>UFRDY:</b> 指示分配浮点堆栈帧时执行的软件是否能够将UsageFault异常置为挂起                                      | 读写 | 0x1 |
| 9     | <b>SPLIMVIOL:</b> 此位为在安全状态间银行化，指示浮点上下文是否违反启用延迟状态保护时的堆栈指针限制SPLIMVIOL修改延迟浮点状态保护行为         | 读写 | 0x0 |
| 8     | <b>MONRDY:</b> 指示分配浮点堆栈帧时执行的软件是否能够将DebugMonitor异常置为挂起                                   | 读写 | 0x0 |
| 7     | <b>SFRDY:</b> 指示分配浮点堆栈帧时执行的软件是否能够将SecureFault异常置为挂起该位仅存在于寄存器的安全版本中，从非安全状态访问时表现为RA Z/WI。 | 读写 | 0x0 |

| 位 | 描述                                                                                   | 类型 | 复位  |
|---|--------------------------------------------------------------------------------------|----|-----|
| 6 | <b>BFRDY</b> : 指示在处理器分配浮点堆栈帧时，执行的软件是否能够将总线错误异常置为待处理状态。                               | 读写 | 0x1 |
| 5 | <b>MMRDY</b> : 指示在处理器分配浮点堆栈帧时，执行的软件是否能够将内存管理异常置为待处理状态。                               | 读写 | 0x1 |
| 4 | <b>HFRDY</b> : 指示在处理器分配浮点堆栈帧时，执行的软件是否能够将硬件故障异常置为待处理状态。                               | 读写 | 0x1 |
| 3 | <b>THREAD</b> : 指示处理器在分配浮点堆栈帧时的运行模式。                                                 | 读写 | 0x0 |
| 2 | <b>S</b> : 浮点上下文的安全状态。该位仅存在于寄存器的安全版本中，从非安全状态访问时表现为RAZ/WI。每当启用延迟状态保存或执行浮点指令时，该位都会被更新。 | 读写 | 0x0 |
| 1 | <b>USER</b> : 指示在处理器分配浮点堆栈帧时执行软件的特权级别。                                               | 读写 | 0x1 |
| 0 | <b>LSPACT</b> : 指示是否启用了浮点状态的惰性保存                                                     | 读写 | 0x0 |

## M33: FPCAR 寄存器

偏移: 0x0ef38

### 描述

存储异常堆栈帧上未使用的浮点寄存器空间位置

表 259. FPCAR 寄存器

| 位    | 描述                         | 类型 | 复位         |
|------|----------------------------|----|------------|
| 31:3 | 地址：异常堆栈帧上为未使用的浮点寄存器空间分配的位置 | 读写 | 0x00000000 |
| 2:0  | 保留。                        | -  | -          |

## M33: FPSCR 寄存器

偏移: 0x0ef3c

### 描述

保存处理器在创建新浮点上下文时赋予 FPSCR 的浮点状态控制数据的默认值

表 260. FPSCR 寄存器

| 位     | 描述                              | 类型 | 复位  |
|-------|---------------------------------|----|-----|
| 31:27 | 保留。                             | -  | -   |
| 26    | <b>AHP</b> : FPSCR.AHP 的默认值     | 读写 | 0x0 |
| 25    | <b>DN</b> : FPSCR.DN 的默认值       | 读写 | 0x0 |
| 24    | <b>FZ</b> : FPSCR.FZ 的默认值       | 读写 | 0x0 |
| 23:22 | <b>RMODE</b> : FPSCR.RMode 的默认值 | 读写 | 0x0 |
| 21:0  | 保留。                             | -  | -   |

## M33: MVFR0 寄存器

偏移: 0x0ef40

### 描述

描述浮点扩展所提供的特性

表 261. MVFR0 寄存器

| 位     | 描述                            | 类型 | 复位  |
|-------|-------------------------------|----|-----|
| 31:28 | <b>FPROUND:</b> 指示浮点扩展支持的舍入模式 | 只读 | 0x6 |
| 27:24 | 保留。                           | -  | -   |
| 23:20 | <b>FPSQRT:</b> 表示支持浮点平方根操作    | 只读 | 0x5 |
| 19:16 | <b>FPDIVIDE:</b> 表示支持浮点除法操作   | 只读 | 0x4 |
| 15:12 | 保留。                           | -  | -   |
| 11:8  | <b>FPDP:</b> 表示支持浮点双精度操作      | 只读 | 0x6 |
| 7:4   | <b>FPSP:</b> 表示支持浮点单精度操作      | 只读 | 0x0 |
| 3:0   | <b>SIMDREG:</b> 表示浮点寄存器文件的大小  | 只读 | 0x1 |

## M33: MVFR1 寄存器

偏移: 0x0ef44

### 描述

描述浮点扩展所提供的特性

表 262. MVFR1 寄存器

| 位     | 描述                               | 类型 | 复位  |
|-------|----------------------------------|----|-----|
| 31:28 | <b>FMAC:</b> 表示浮点扩展是否实现融合乘加指令    | 只读 | 0x8 |
| 27:24 | <b>FPHP:</b> 表示浮点扩展是否实现半精度浮点转换指令 | 只读 | 0x5 |
| 23:8  | 保留。                              | -  | -   |
| 7:4   | <b>FPDNAN:</b> 表示浮点硬件实现是否支持NaN传播 | 只读 | 0x8 |
| 3:0   | <b>FPFTZ:</b> 表示次正规数是否总被清零       | 只读 | 0x9 |

## M33: MVFR2 寄存器

偏移: 0x0ef48

### 描述

描述浮点扩展所提供的特性

表 263. MVFR2 寄存器

| 位    | 描述                        | 类型 | 复位  |
|------|---------------------------|----|-----|
| 31:8 | 保留。                       | -  | -   |
| 7:4  | <b>FPMISC:</b> 表示支持浮点杂项功能 | 只读 | 0x6 |
| 3:0  | 保留。                       | -  | -   |

## M33: DDEVARCH 寄存器

偏移: 0x0efbc

**描述**

为SCS提供CoreSight发现信息

表264. DDEVARCH寄存器

| 位     | 描述                                                                                | 类型 | 复位    |
|-------|-----------------------------------------------------------------------------------|----|-------|
| 31:21 | <b>ARCHITECT</b> : 定义组件的架构商。位[31:28]为JEP106连续码（JEP106银行ID减1），位[27:21]为JEP106 ID码。 | 只读 | 0x23b |
| 20    | <b>PRESENT</b> : 定义DEVARCH寄存器的存在                                                  | 只读 | 0x1   |
| 19:16 | <b>REVISION</b> : 定义组件的架构修订版                                                      | 只读 | 0x0   |
| 15:12 | <b>ARCHVER</b> : 定义组件的架构版本                                                        | 只读 | 0x2   |
| 11:0  | <b>ARCHPART</b> : 定义组件的架构                                                         | 只读 | 0xa04 |

**M33: DDEVTYPE 寄存器**

偏移: 0x0efcc

**描述**

为SCS提供CoreSight发现信息

表265. DDEVTYPE寄存器

| 位    | 描述                 | 类型 | 复位  |
|------|--------------------|----|-----|
| 31:8 | 保留。                | -  | -   |
| 7:4  | <b>SUB</b> : 组件子类型 | 只读 | 0x0 |
| 3:0  | 主版本: CoreSight 主类型 | 只读 | 0x0 |

**M33: DPIDR4 寄存器**

偏移: 0x0efd0

**描述**

为SCS提供CoreSight发现信息

表266. DPIDR4寄存器

| 位    | 描述                             | 类型 | 复位  |
|------|--------------------------------|----|-----|
| 31:8 | 保留。                            | -  | -   |
| 7:4  | <b>SIZE</b> : 详见CoreSight架构规范  | 只读 | 0x0 |
| 3:0  | <b>DES_2</b> : 详见CoreSight架构规范 | 只读 | 0x4 |

**M33: DPIDR5 寄存器**

偏移: 0x0efd4

**描述**

为SCS提供CoreSight发现信息

表267. DPIDR5寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33: DPIDR6 寄存器**

偏移: 0x0efd8

**描述**

为SCS提供CoreSight发现信息

表 268. DPIDR7  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33：DPIDR7 寄存器**

偏移: 0x0efdc

**描述**

为SCS提供CoreSight发现信息

表 269. DPIDR7  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

**M33：DPIDR0 寄存器**

偏移: 0x0efe0

**描述**

为SCS提供CoreSight发现信息

表 270. DPIDR0  
寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PART_0</b> : 参见 CoreSight 架构规范 | 只读 | 0x21 |

**M33：DPIDR1 寄存器**

偏移: 0x0efe4

**描述**

为SCS提供CoreSight发现信息

表 271. DPIDR1  
寄存器

| 位    | 描述                                | 类型 | 复位  |
|------|-----------------------------------|----|-----|
| 31:8 | 保留。                               | -  | -   |
| 7:4  | <b>DES_0</b> : 参见 CoreSight 架构规范  | 只读 | 0xb |
| 3:0  | <b>PART_1</b> : 参见 CoreSight 架构规范 | 只读 | 0xd |

**M33：DPIDR2 寄存器**

偏移: 0x0efe8

**描述**

为SCS提供CoreSight发现信息

表 272. DPIDR2  
寄存器

| 位    | 描述                             | 类型 | 复位  |
|------|--------------------------------|----|-----|
| 31:8 | 保留。                            | -  | -   |
| 7:4  | 版本: 详见CoreSight架构规范            | 只读 | 0x0 |
| 3    | <b>JEDEC</b> : 详见CoreSight架构规范 | 只读 | 0x1 |
| 2:0  | <b>DES_1</b> : 详见CoreSight架构规范 | 只读 | 0x3 |

## M33: DPIDR3寄存器

偏移: 0x0efec

### 描述

为SCS提供CoreSight发现信息

表273. DPIDR3  
寄存器

| 位    | 描述                           | 类型 | 复位  |
|------|------------------------------|----|-----|
| 31:8 | 保留。                          | -  | -   |
| 7:4  | 修订: 详见CoreSight架构规范          | 只读 | 0x0 |
| 3:0  | <b>CMOD:</b> 详见CoreSight架构规范 | 只读 | 0x0 |

## M33: DCIDR0寄存器

偏移: 0x0eff0

### 描述

为SCS提供CoreSight发现信息

表274. DCIDR0  
寄存器

| 位    | 描述                              | 类型 | 复位   |
|------|---------------------------------|----|------|
| 31:8 | 保留。                             | -  | -    |
| 7:0  | <b>PRMBL_0:</b> 详见CoreSight架构规范 | 只读 | 0x0d |

## M33: DCIDR1寄存器

偏移: 0x0eff4

### 说明

为SCS提供CoreSight发现信息

表275. DCIDR1  
寄存器

| 位    | 描述                                | 类型 | 复位  |
|------|-----------------------------------|----|-----|
| 31:8 | 保留。                               | -  | -   |
| 7:4  | 类别: 参见 CoreSight 架构规范             | 只读 | 0x9 |
| 3:0  | <b>PRMBL_1:</b> 参见 CoreSight 架构规范 | 只读 | 0x0 |

## M33: DCIDR2寄存器

偏移: 0x0eff8

### 说明

为SCS提供CoreSight发现信息

表276. DCIDR2  
寄存器

| 位    | 描述                                | 类型 | 复位   |
|------|-----------------------------------|----|------|
| 31:8 | 保留。                               | -  | -    |
| 7:0  | <b>PRMBL_2:</b> 参见 CoreSight 架构规范 | 只读 | 0x05 |

## M33: DCIDR3寄存器

偏移: 0x0effc

### 说明

为SCS提供CoreSight发现信息

表277. DCIDR3  
寄存器

| 位    | 描述                                 | 类型 | 复位   |
|------|------------------------------------|----|------|
| 31:8 | 保留。                                | -  | -    |
| 7:0  | <b>PRMBL_3</b> : 参见 CoreSight 架构规范 | 只读 | 0xb1 |

## M33: TRCPRGCTRL寄存器

偏移: 0x41004

### 说明

编程控制寄存器

表278.  
TRCPRGCTRL寄存器

| 位    | 描述                 | 类型 | 复位  |
|------|--------------------|----|-----|
| 31:1 | 保留。                | -  | -   |
| 0    | <b>EN</b> : 跟踪单元使能 | 读写 | 0x0 |

## M33: TRCSTATR寄存器

偏移: 0x4100c

### 说明

TRCSTATR 指示 ETM-Teal 状态

表279. TRCSTATR  
寄存器

| 位    | 描述                                      | 类型 | 复位  |
|------|-----------------------------------------|----|-----|
| 31:2 | 保留。                                     | -  | -   |
| 1    | <b>PMSTABLE</b> : 指示ETM-Teal寄存器是否稳定且可读取 | 只读 | 0x0 |
| 0    | <b>IDLE</b> : 指示跟踪单元处于空闲状态              | 只读 | 0x0 |

## M33: TRCONFIGR寄存器

偏移: 0x41010

### 描述

TRCONFIGR 设置跟踪单元的基本跟踪选项

表280.  
TRCONFIGR寄存器

| 位     | 描述                      | 类型 | 复位   |
|-------|-------------------------|----|------|
| 31:13 | 保留。                     | -  | -    |
| 12    | <b>RS</b> : 返回栈使能       | 读写 | 0x0  |
| 11    | <b>TS</b> : 全局时间戳追踪     | 读写 | 0x0  |
| 10:5  | <b>COND</b> : 条件指令追踪    | 读写 | 0x00 |
| 4     | <b>CCI</b> : 指令追踪中的周期计数 | 读写 | 0x0  |
| 3     | <b>BB</b> : 分支广播模式      | 读写 | 0x0  |
| 2:0   | 保留。                     | -  | -    |

## M33: TRCEVENTCTL0R寄存器

偏移: 0x41020

### 描述

TRCEVENTCTL0R 控制跟踪流中的事件跟踪。事件还驱动ETM-Teal

外部输出。

表281。  
TRCEVENTCTL0R  
寄存器

| 位     | 描述                                                                                                           | 类型 | 复位  |
|-------|--------------------------------------------------------------------------------------------------------------|----|-----|
| 31:16 | 保留。                                                                                                          | -  | -   |
| 15    | <b>TYPE1</b> : 选择事件1的资源类型                                                                                    | 读写 | 0x0 |
| 14:11 | 保留。                                                                                                          | -  | -   |
| 10:8  | <b>SEL1</b> : 根据TYPE1的值选择资源编号：当TYPE1为0时，从SEL1[2:0]定义的0-15单个资源中选择。<br>当TYPE1为1时，从SEL1[2:0]定义的0-7布尔组合资源对中进行选择。 | 读写 | 0x0 |
| 7     | <b>TYPE0</b> : 选择事件0的资源类型                                                                                    | 读写 | 0x0 |
| 6:3   | 保留。                                                                                                          | -  | -   |
| 2:0   | <b>SEL0</b> : 选择资源编号，基于TYPE0的值；当TYPE1为0时，选择由SEL0[2:0]定义的0至15中单个资源。<br>当TYPE1为1时，选择由SEL0[2:0]定义的0至7中布尔组合资源对。  | 读写 | 0x0 |

## M33: TRCEVENTCTL1R寄存器

偏移: 0x41024

### 描述

TRCEVENTCTL1R 控制 TRCEVENTCTL0R 所选事件的行为

表282。  
TRCEVENTCTL1R  
寄存器

| 位     | 描述                                                 | 类型 | 复位  |
|-------|----------------------------------------------------|----|-----|
| 31:13 | 保留。                                                | -  | -   |
| 12    | <b>LPOVERRIDE</b> : 低功耗状态行为覆盖                      | 读写 | 0x0 |
| 11    | <b>ATB</b> : 启用ATB                                 | 读写 | 0x0 |
| 10:2  | 保留。                                                | -  | -   |
| 1     | <b>INSTEN1</b> : 每个事件一位，用于使能在选定事件发生时，在指令跟踪流中生成事件元素 | 读写 | 0x0 |
| 0     | <b>INSTENO</b> : 每个事件一位，用于使能在选定事件发生时，在指令跟踪流中生成事件元素 | 读写 | 0x0 |

## M33: TRCSTALLCTRL寄存器

偏移: 0x4102c

### 描述

TRCSTALLCTRL 使能 ETM-Teal 在 FIFO 超过设定阈值时暂停处理器，以降低溢出风险

表283。  
TRCSTALLCTRL  
寄存器

| 位     | 描述                               | 类型 | 复位  |
|-------|----------------------------------|----|-----|
| 31:11 | 保留。                              | -  | -   |
| 10    | <b>INSPRIORITY</b> : 保留，RES0     | 只读 | 0x0 |
| 9     | 保留。                              | -  | -   |
| 8     | <b>ISTALL</b> : 根据指令跟踪缓冲区空间阻塞处理器 | 读写 | 0x0 |
| 7:4   | 保留。                              | -  | -   |

| 位   | 描述                                                                            | 类型 | 复位  |
|-----|-------------------------------------------------------------------------------|----|-----|
| 3:2 | <b>LEVEL:</b> 阈值，达到该值后阻塞开始生效。该设置提供四个级别。<br>此级别可调节，以优化阻塞带来的影响程度，同时平衡FIFO溢出的风险。 | 读写 | 0x0 |
| 1:0 | 保留。                                                                           | -  | -   |

## M33: TRCTSCLR 寄存器

偏移: 0x41030

### 描述

TRCTSCLR 控制全局时间戳的插入，以导入跟踪流中。时间戳始终插入到指令跟踪流中。

表 284。  
TRCTSCLR 寄存器

| 位    | 描述                                                                                                         | 类型 | 复位  |
|------|------------------------------------------------------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                                                                        | -  | -   |
| 7    | <b>TYPE0:</b> 选择事件0的资源类型                                                                                   | 读写 | 0x0 |
| 6:2  | 保留。                                                                                                        | -  | -   |
| 1:0  | <b>SEL0:</b> 选择资源编号，基于TYPE0的值；当TYPE1为0时，选择由SEL0[2:0]定义的0至15中单个资源。<br>当TYPE1为1时，选择由SEL0[2:0]定义的0至7中布尔组合资源对。 | 读写 | 0x0 |

## M33: TRCSYNCPR 寄存器

偏移: 0x41034

### 描述

TRCSYNCPR 指定跟踪流的同步周期。TRCSYNCPR 定义请求跟踪同步之间的跟踪字节数。该值始终为二的幂。

表 285。  
TRCSYNCPR 寄存器

| 位    | 描述                                                                                                                                                                                      | 类型 | 复位   |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:5 | 保留。                                                                                                                                                                                     | -  | -    |
| 4:0  | <b>PERIOD:</b> 定义指令流生成的字节数中，用于跟踪同步请求的字节总数。字节数为 $2N$ ，其中 $N$ 为此字段的值：<br>- 该字段为零时禁用周期性跟踪同步请求，但不影响其他跟踪同步请求的启用。<br>- 除零以外，可编程的最小值为8，提供最小的跟踪同步周期为256字节。<br>- 最大值为20，提供最大跟踪同步周期为 $2^{20}$ 字节。 | 只读 | 0x0a |

## M33: TRCCCCTRLR 寄存器

偏移: 0x41038

### 描述

TRCCCCTRLR 设定指令跟踪周期计数的阈值。该阈值表示周期计数跟踪包之间的最小间隔。

表 286。  
TRCCCCTRLR 寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:12 | 保留。 | -  | -  |

| 位    | 描述                            | 类型 | 复位    |
|------|-------------------------------|----|-------|
| 11:0 | <b>THRESHOLD</b> : 指令追踪周期计数阈值 | 读写 | 0x000 |

## M33: TRCVICTLR 寄存器

偏移: 0x41080

### 描述

TRCVICTLR 控制指令跟踪过滤。

表 287. TRCVICTLR  
寄存器

| 位     | 描述                                                                                                          | 类型 | 复位  |
|-------|-------------------------------------------------------------------------------------------------------------|----|-----|
| 31:20 | 保留。                                                                                                         | -  | -   |
| 19    | <b>EXLEVEL_S3</b> : 在安全状态下，每个位控制对应异常级别是否启用指令追踪                                                              | 读写 | 0x0 |
| 18:17 | 保留。                                                                                                         | -  | -   |
| 16    | <b>EXLEVEL_S0</b> : 在安全状态下，每个位控制对应异常级别是否启用指令追踪                                                              | 读写 | 0x0 |
| 15:12 | 保留。                                                                                                         | -  | -   |
| 11    | <b>TRCERR</b> : 选择系统错误异常是否必须始终追踪                                                                            | 读写 | 0x0 |
| 10    | <b>TRCRESET</b> : 选择复位异常是否必须始终追踪                                                                            | 读写 | 0x0 |
| 9     | <b>SSSTATUS</b> : 指示启动/停止逻辑的当前状态                                                                            | 读写 | 0x0 |
| 8     | 保留。                                                                                                         | -  | -   |
| 7     | <b>TYPE0</b> : 选择事件0的资源类型                                                                                   | 读写 | 0x0 |
| 6:2   | 保留。                                                                                                         | -  | -   |
| 1:0   | <b>SEL0</b> : 选择资源编号，基于TYPE0的值；当TYPE1为0时，选择由SEL0[2:0]定义的0至15中单个资源。<br>当TYPE1为1时，选择由SEL0[2:0]定义的0至7中布尔组合资源对。 | 读写 | 0x0 |

## M33: TRCCNTRLDVR0 寄存器

偏移: 0x41140

### 描述

TRCCNTRLDVR0 定义了简化功能计数器的重载值。

表 288.  
TRCCNTRLDVR0  
寄存器

| 位     | 描述                                  | 类型 | 复位     |
|-------|-------------------------------------|----|--------|
| 31:16 | 保留。                                 | -  | -      |
| 15:0  | 数值：定义计数器的重载值。每当发生重载事件时，此值会被加载到计数器中。 | 读写 | 0x0000 |

## M33: TRCIDR8 寄存器

偏移: 0x41180

### 描述

TRCIDR8

表 289。TRCIDR8  
寄存器

| 位    | 描述                           | 类型 | 复位         |
|------|------------------------------|----|------------|
| 31:0 | <b>MAXSPEC</b> : 读取为 `ImpDef | 只读 | 0x00000000 |

## M33：TRCIDR9 寄存器

偏移: 0x41184

### 描述

TRCIDR9

表 290。TRCIDR9  
寄存器

| 位    | 描述                            | 类型 | 复位         |
|------|-------------------------------|----|------------|
| 31:0 | <b>NUMP0KEY</b> : 读取为 `ImpDef | 只读 | 0x00000000 |

## M33：TRCIDR10 寄存器

偏移: 0x41188

### 描述

TRCIDR10

表 291。TRCIDR10  
寄存器

| 位    | 描述                            | 类型 | 复位         |
|------|-------------------------------|----|------------|
| 31:0 | <b>NUMP1KEY</b> : 读取为 `ImpDef | 只读 | 0x00000000 |

## M33：TRCIDR11 寄存器

偏移: 0x4118c

### 描述

TRCIDR11

表 292。TRCIDR11  
寄存器

| 位    | 描述                            | 类型 | 复位         |
|------|-------------------------------|----|------------|
| 31:0 | <b>NUMP1SPC</b> : 读取为 `ImpDef | 只读 | 0x00000000 |

## M33：TRCIDR12 寄存器

偏移: 0x41190

### 描述

TRCIDR12

表 293。TRCIDR12  
寄存器

| 位    | 描述                              | 类型 | 复位         |
|------|---------------------------------|----|------------|
| 31:0 | <b>NUMCONDKEY</b> : 读取为 `ImpDef | 只读 | 0x00000001 |

## M33：TRCIDR13 寄存器

偏移: 0x41194

### 描述

TRCIDR13

表 294. TRCIDR13  
寄存器

| 位    | 描述                              | 类型 | 复位         |
|------|---------------------------------|----|------------|
| 31:0 | <b>NUMCONDSPC</b> : 读取为 `ImpDef | 只读 | 0x00000000 |

## M33: TRCIMSPEC 寄存器

偏移: 0x411c0

### 描述

TRCIMSPEC显示任何特定实现功能的存在，并启用所有提供的功能。

表 295.  
TRCIMSPEC 寄存器

| 位    | 描述                        | 类型 | 复位  |
|------|---------------------------|----|-----|
| 31:4 | 保留。                       | -  | -   |
| 3:0  | <b>SUPPORT</b> : 保留， RES0 | 只读 | 0x0 |

## M33: TRCIDR0 寄存器

偏移: 0x411e0

### 描述

TRCIDR0

表 296. TRCIDR0  
寄存器

| 位     | 描述                              | 类型 | 复位   |
|-------|---------------------------------|----|------|
| 31:30 | 保留。                             | -  | -    |
| 29    | <b>COMMOPT</b> : 读取值为 `ImpDef   | 只读 | 0x1  |
| 28:24 | <b>TSSIZE</b> : 读取值为 `ImpDef    | 只读 | 0x08 |
| 23:18 | 保留。                             | -  | -    |
| 17    | <b>TRCEXDATA</b> : 读取值为 `ImpDef | 只读 | 0x0  |
| 16:15 | <b>QSUPP</b> : 读取值为 `ImpDef     | 只读 | 0x0  |
| 14    | <b>QFILT</b> : 读取值为 `ImpDef     | 只读 | 0x0  |
| 13:12 | <b>CONDTYPE</b> : 读取值为 `ImpDef  | 只读 | 0x0  |
| 11:10 | <b>NUMEVENT</b> : 读取值为 `ImpDef  | 只读 | 0x1  |
| 9     | <b>RETSTACK</b> : 读取值为 `ImpDef  | 只读 | 0x1  |
| 8     | 保留。                             | -  | -    |
| 7     | <b>TRCCCI</b> : 读取值为 `ImpDef    | 只读 | 0x1  |
| 6     | <b>TRCCOND</b> : 读取值为 `ImpDef   | 只读 | 0x1  |
| 5     | <b>TRCBB</b> : 读取值为 `ImpDef     | 只读 | 0x1  |
| 4:3   | <b>TRCDATA</b> : 读取值为 `ImpDef   | 只读 | 0x0  |
| 2:1   | <b>INSTPO</b> : 读取值为 `ImpDef    | 只读 | 0x0  |
| 0     | <b>RES1</b> : 保留， RES1          | 只读 | 0x1  |

## M33: TRCIDR1 寄存器

偏移: 0x411e4

### 描述

TRCIDR1

表297. TRC IDR1  
寄存器

| 位     | 描述                            | 类型 | 复位   |
|-------|-------------------------------|----|------|
| 31:24 | 设计者: 读作 `ImpDef               | 只读 | 0x41 |
| 23:16 | 保留。                           | -  | -    |
| 15:12 | <b>RES1</b> : 保留, RES1        | 只读 | 0xf  |
| 11:8  | <b>TRCARCHMAJ</b> : 读作 0b0100 | 只读 | 0x4  |
| 7:4   | <b>TRCARCHMIN</b> : 读作 0b0000 | 只读 | 0x2  |
| 3:0   | 版本: 读作 `ImpDef                | 只读 | 0x1  |

## M33: TRC IDR2 寄存器

偏移: 0x411e8

### 描述

TRC IDR2

表298. TRC IDR2  
寄存器

| 位     | 描述                           | 类型 | 复位   |
|-------|------------------------------|----|------|
| 31:29 | 保留。                          | -  | -    |
| 28:25 | <b>CCSIZE</b> : 读作 `ImpDef   | 只读 | 0x0  |
| 24:20 | <b>DVSIZE</b> : 读作 `ImpDef   | 只读 | 0x00 |
| 19:15 | <b>DASIZE</b> : 读作 `ImpDef   | 只读 | 0x00 |
| 14:10 | <b>VMIDSIZE</b> : 读作 `ImpDef | 只读 | 0x00 |
| 9:5   | <b>CIDSIZE</b> : 读作 `ImpDef  | 只读 | 0x00 |
| 4:0   | <b>IASIZE</b> : 读作 `ImpDef   | 只读 | 0x04 |

## M33: TRC IDR3 寄存器

偏移: 0x411ec

### 描述

TRC IDR3

表299. TRC IDR3  
寄存器

| 位     | 描述                              | 类型 | 复位    |
|-------|---------------------------------|----|-------|
| 31    | 无溢出: 读作 `ImpDef                 | 只读 | 0x0   |
| 30:28 | <b>NUMPROC</b> : 读取为 `ImpDef    | 只读 | 0x0   |
| 27    | <b>SYSSTALL</b> : 读取为 `ImpDef   | 只读 | 0x1   |
| 26    | <b>STALLCTL</b> : 读取为 `ImpDef   | 只读 | 0x1   |
| 25    | <b>SYNCPR</b> : 读取为 `ImpDef     | 只读 | 0x1   |
| 24    | <b>TRCERR</b> : 读取为 `ImpDef     | 只读 | 0x1   |
| 23:20 | <b>EXLEVEL_NS</b> : 读取为 `ImpDef | 只读 | 0x0   |
| 19:16 | <b>EXLEVEL_S</b> : 读取为 `ImpDef  | 只读 | 0x9   |
| 15:12 | 保留。                             | -  | -     |
| 11:0  | <b>CCITMIN</b> : 读取为 `ImpDef    | 只读 | 0x004 |

## M33: TRC IDR4 寄存器

偏移量: 0x411f0

**描述**

TRCIDR4

表300. TRCIDR4  
寄存器

| 位     | 描述                               | 类型 | 复位  |
|-------|----------------------------------|----|-----|
| 31:28 | <b>NUMVMIDC</b> : 读取为 `ImpDef    | 只读 | 0x0 |
| 27:24 | <b>NUMCIDC</b> : 读取为 `ImpDef     | 只读 | 0x0 |
| 23:20 | <b>NUMSSCC</b> : 读取为 `ImpDef     | 只读 | 0x1 |
| 19:16 | <b>NUMRSPAIR</b> : 读取为 `ImpDef   | 只读 | 0x1 |
| 15:12 | <b>NUMPC</b> : 读取为 `ImpDef       | 只读 | 0x4 |
| 11:9  | 保留。                              | -  | -   |
| 8     | <b>SUPPDAC</b> : 读取为 `ImpDef     | 只读 | 0x0 |
| 7:4   | <b>NUMDVC</b> : 读取为 `ImpDef      | 只读 | 0x0 |
| 3:0   | <b>NUMACPAIRS</b> : 读取值为 `ImpDef | 只读 | 0x0 |

**M33: TRCIDR5 寄存器**

偏移: 0x411f4

**描述**

TRCIDR5

表 301. TRCIDR5  
寄存器

| 位     | 描述                                | 类型 | 复位    |
|-------|-----------------------------------|----|-------|
| 31    | <b>REDFUNCNTR</b> : 读取值为 `ImpDef  | 只读 | 0x1   |
| 30:28 | <b>NUMCNTR</b> : 读取值为 `ImpDef     | 只读 | 0x1   |
| 27:25 | <b>NUMSEQSTATE</b> : 读取值为 `ImpDef | 只读 | 0x0   |
| 24    | 保留。                               | -  | -     |
| 23    | <b>LPOVERRIDE</b> : 读取值为 `ImpDef  | 只读 | 0x1   |
| 22    | <b>ATBTRIG</b> : 读取值为 `ImpDef     | 只读 | 0x1   |
| 21:16 | <b>TRACEIDSIZE</b> : 读取值为 0x07    | 只读 | 0x07  |
| 15:12 | 保留。                               | -  | -     |
| 11:9  | <b>NUMEXTINSEL</b> : 读取值为 `ImpDef | 只读 | 0x0   |
| 8:0   | <b>NUMEXTIN</b> : 读取值为 `ImpDef    | 只读 | 0x004 |

**M33: TRCIDR6 寄存器**

偏移: 0x411f8

**描述**

TRCIDR6

表 302. TRC IDR6  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33：TRC IDR7 寄存器

偏移: 0x411fc

### 描述

TRC IDR7

表 303. TRC IDR7  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33：TRCRSCTRLR2寄存器

偏移: 0x41208

### 说明

TRCRSCTRLR 控制跟踪资源

表 304.  
TRCRSCTRLR2寄存器

| 位     | 描述                                                 | 类型 | 复位   |
|-------|----------------------------------------------------|----|------|
| 31:22 | 保留。                                                | -  | -    |
| 21    | <b>PAIRINV</b> : 反转组合资源对的结果。该位仅在资源选择器对的低位寄存器中实现。   | 读写 | 0x0  |
| 20    | <b>INV</b> : 反转所选资源                                | 读写 | 0x0  |
| 19    | 保留。                                                | -  | -    |
| 18:16 | <b>GROUP</b> : 选择资源组                               | 读写 | 0x0  |
| 15:8  | 保留。                                                | -  | -    |
| 7:0   | <b>SELECT</b> : 从指定组中选择一个或多个资源。每个资源组中的资源对应一位。<br>◦ | 读写 | 0x00 |

## M33：TRCRSCTRLR3寄存器

偏移: 0x4120c

### 说明

TRCRSCTRLR 控制跟踪资源

表 305.  
TRCRSCTRLR3寄存器

| 位     | 描述                                               | 类型 | 复位  |
|-------|--------------------------------------------------|----|-----|
| 31:22 | 保留。                                              | -  | -   |
| 21    | <b>PAIRINV</b> : 反转组合资源对的结果。该位仅在资源选择器对的低位寄存器中实现。 | 读写 | 0x0 |
| 20    | <b>INV</b> : 反转所选资源                              | 读写 | 0x0 |
| 19    | 保留。                                              | -  | -   |
| 18:16 | <b>GROUP</b> : 选择资源组                             | 读写 | 0x0 |
| 15:8  | 保留。                                              | -  | -   |

| 位   | 描述                                                 | 类型 | 复位   |
|-----|----------------------------------------------------|----|------|
| 7:0 | <b>SELECT</b> : 从指定组中选择一个或多个资源。每个资源组中的资源对应一位。<br>。 | 读写 | 0x00 |

## M33: TRCSSCSR 寄存器

偏移: 0x412a0

### 说明

控制对应的单次比较器资源

表306. TRCSSCSR 寄存器

| 位    | 描述                                                                       | 类型 | 复位  |
|------|--------------------------------------------------------------------------|----|-----|
| 31   | <b>STATUS</b> : 单次触发状态位。指示由 TRCSSCCRn.SAC 或 TRCSSCCRn.A RC 选择的任一比较器是否匹配。 | 读写 | 0x0 |
| 30:4 | 保留。                                                                      | -  | -   |
| 3    | <b>PC</b> : 保留, RES1                                                     | 只读 | 0x0 |
| 2    | <b>DV</b> : 保留, RES0                                                     | 只读 | 0x0 |
| 1    | <b>DA</b> : 保留, RES0                                                     | 只读 | 0x0 |
| 0    | <b>INST</b> : 保留, RES0                                                   | 只读 | 0x0 |

## M33: TRCSSPCICR 寄存器

偏移量: 0x412c0

### 说明

选择单次控制用的PE比较器输入

表 307.  
TRCSSPCICR 寄存器

| 位    | 描述                                                                                                                                  | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                                                                                                 | -  | -   |
| 3:0  | <b>PC</b> : 选择一个或多个 PE 比较器输入以实现单次触发控制。<br>TRCIDR4.NUMPC 定义 PC 字段的大小，每个已实现的 PE 比较器输入对应 1 位。例如，若 bit[1] == 1，则选择 PE 比较器输入 1 用于单次触发控制。 | 读写 | 0x0 |

## M33: TRCPDCR 寄存器

偏移量: 0x41310

### 说明

请求系统为追踪单元供电

表 308. TRCPDCR  
寄存器

| 位    | 描述                 | 类型 | 复位  |
|------|--------------------|----|-----|
| 31:4 | 保留。                | -  | -   |
| 3    | <b>PU</b> : 上电请求位: | 读写 | 0x0 |
| 2:0  | 保留。                | -  | -   |

## M33: TRCPDSR 寄存器

偏移: 0x41314

**描述**

返回关于追踪单元的以下信息： - 操作系统锁定状态- 核心电源域状态- 电源中断状态

表309. TRCPDSR  
寄存器

| 位    | 描述                                       | 类型 | 复位  |
|------|------------------------------------------|----|-----|
| 31:6 | 保留。                                      | -  | -   |
| 5    | <b>OSLK</b> : 操作系统锁定状态位：                 | 只读 | 0x0 |
| 4:2  | 保留。                                      | -  | -   |
| 1    | <b>STICKYPD</b> : 粘滞断电状态位。指示跟踪寄存器状态是否有效： | 只读 | 0x1 |
| 0    | <b>POWER</b> : 电源状态位：                    | 只读 | 0x1 |

**M33: TRCITATBIDR寄存器**

偏移: 0x41ee4

**描述**

追踪集成ATB识别寄存器

表310。  
TRCITATBIDR寄存器

| 位    | 描述               | 类型 | 复位   |
|------|------------------|----|------|
| 31:7 | 保留。              | -  | -    |
| 6:0  | <b>ID</b> : 跟踪ID | 读写 | 0x00 |

**M33: TRCITIATBINR寄存器**

偏移: 0x41ef4

**描述**

跟踪集成指令ATB输入寄存器

表311。  
TRCITIATBINR  
寄存器

| 位    | 描述                                   | 类型 | 复位  |
|------|--------------------------------------|----|-----|
| 31:2 | 保留。                                  | -  | -   |
| 1    | <b>AFVALIDM</b> : 集成模式指令 AFVALIDM 输入 | 读写 | 0x0 |
| 0    | <b>ATREADYM</b> : 集成模式指令 ATREADYM 输入 | 读写 | 0x0 |

**M33: TRCITIATBOUTR寄存器**

偏移: 0x41efc

**说明**

跟踪集成指令ATB输出寄存器

表 312.  
TRCITIATBOUTR  
寄存器

| 位    | 描述                                 | 类型 | 复位  |
|------|------------------------------------|----|-----|
| 31:2 | 保留。                                | -  | -   |
| 1    | <b>AFREADY</b> : 集成模式指令 AFREADY 输出 | 读写 | 0x0 |
| 0    | <b>ATVALID</b> : 集成模式指令 ATVALID 输出 | 读写 | 0x0 |

**M33: TRCCLAIMSET 寄存器**

偏移: 0x41fa0

**说明**

声明标记设置寄存器

表 313.  
TRCCLAIMSET  
寄存器

| 位    | 描述                         | 类型 | 复位  |
|------|----------------------------|----|-----|
| 31:4 | 保留。                        | -  | -   |
| 3    | <b>SET3:</b> 对这些位之一写入以下值时： | 读写 | 0x1 |
| 2    | <b>SET2:</b> 对这些位之一写入以下值时： | 读写 | 0x1 |
| 1    | <b>SET1:</b> 对这些位之一写入以下值时： | 读写 | 0x1 |
| 0    | <b>SET0:</b> 对这些位之一写入以下值时： | 读写 | 0x1 |

**M33: TRCCLAIMCLR 寄存器**

偏移: 0x41fa4

**描述**

声明标签清除寄存器

表 314.  
TRCCLAIMCLR  
寄存器

| 位    | 描述                         | 类型 | 复位  |
|------|----------------------------|----|-----|
| 31:4 | 保留。                        | -  | -   |
| 3    | <b>CLR3:</b> 当以以下值写入该位之一时： | 读写 | 0x0 |
| 2    | <b>CLR2:</b> 当以以下值写入该位之一时： | 读写 | 0x0 |
| 1    | <b>CLR1:</b> 当以以下值写入该位之一时： | 读写 | 0x0 |
| 0    | <b>CLR0:</b> 当以以下值写入该位之一时： | 读写 | 0x0 |

**M33: TRCAUTHSTATUS 寄存器**

偏移: 0x41fb8

**描述**

返回跟踪单元所支持的跟踪级别

表 315。  
TRCAUTHSTATUS  
寄存器

| 位    | 描述                                      | 类型 | 复位  |
|------|-----------------------------------------|----|-----|
| 31:8 | 保留。                                     | -  | -   |
| 7:6  | <b>SNID:</b> 指示系统是否启用跟踪单元以支持安全非侵入式调试：   | 只读 | 0x0 |
| 5:4  | <b>SID:</b> 指示跟踪单元是否支持安全侵入式调试：          | 只读 | 0x0 |
| 3:2  | <b>NSNID:</b> 指示系统是否启用跟踪单元以支持非安全非侵入式调试： | 只读 | 0x0 |
| 1:0  | <b>NSID:</b> 指示跟踪单元是否支持非安全侵入式调试：        | 只读 | 0x0 |

**M33: TRCDEVARCH 寄存器**

偏移: 0x41fbc

**描述**

TRCDEVARCH

表 316。  
TRCDEVARCH 寄存器

| 位     | 描述                                  | 类型 | 复位    |
|-------|-------------------------------------|----|-------|
| 31:21 | <b>ARCHITECT:</b> 读取值为0b01000111011 | 只读 | 0x23b |

| 位     | 描述                                     | 类型 | 复位     |
|-------|----------------------------------------|----|--------|
| 20    | <b>PRESENT</b> : 读取值为0b1               | 只读 | 0x1    |
| 19:16 | <b>REVISION</b> : 读取值为0b0000           | 只读 | 0x2    |
| 15:0  | <b>ARCHID</b> : 读取值为0b0100101000010011 | 只读 | 0x4a13 |

## M33: TRCDEVID寄存器

偏移量: 0x41fc8

### 说明

TRCDEVID

表317. TRCDEVID  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: TRCDEVTYPE寄存器

偏移量: 0x41fcc

### 描述

TRCDEVTYPE

表318。  
TRCDEVTYPE寄存器

| 位    | 描述                        | 类型 | 复位  |
|------|---------------------------|----|-----|
| 31:8 | 保留。                       | -  | -   |
| 7:4  | <b>SUB</b> : 读取值为0b0001   | 只读 | 0x1 |
| 3:0  | <b>MAJOR</b> : 读取值为0b0011 | 只读 | 0x3 |

## M33: TRCPIDR4寄存器

偏移量: 0x41fd0

### 描述

TRCPIDR4

表319. TRCPIDR4  
寄存器

| 位    | 描述                        | 类型 | 复位  |
|------|---------------------------|----|-----|
| 31:8 | 保留。                       | -  | -   |
| 7:4  | <b>SIZE</b> : 读取为`ImpDef  | 只读 | 0x0 |
| 3:0  | <b>DES_2</b> : 读取为`ImpDef | 只读 | 0x4 |

## M33: TRCPIDR5寄存器

偏移量: 0x41fd4

### 描述

TRCPIDR5

表320. TRCPIDR5  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33：TRCPIDR6寄存器

偏移量: 0x41fd8

### 描述

TRCPIDR6

表321. TRCPIDR6  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33：TRCPIDR7寄存器

偏移量: 0x41fdc

### 描述

TRCPIDR7

表322. TRCPIDR7  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33：TRCPIDR0寄存器

偏移量: 0x41fe0

### 描述

TRCPIDR0

表323. TRCPIDR0  
寄存器

| 位    | 描述                          | 类型 | 复位   |
|------|-----------------------------|----|------|
| 31:8 | 保留。                         | -  | -    |
| 7:0  | <b>PART_0</b> : 读取为 `ImpDef | 只读 | 0x21 |

## M33：TRCPIDR1寄存器

偏移量: 0x41fe4

### 描述

TRCPIDR1

表324. TRCPIDR1  
寄存器

| 位    | 描述                          | 类型 | 复位  |
|------|-----------------------------|----|-----|
| 31:8 | 保留。                         | -  | -   |
| 7:4  | <b>DES_0</b> : 读取值为 `ImpDef | 只读 | 0xb |
| 3:0  | <b>PART_0</b> : 读取为 `ImpDef | 只读 | 0xd |

## M33：TRCPIDR2 寄存器

偏移: 0x41fe8

### 描述

TRCPIDR2

表325. TRCPIDR2  
寄存器

| 位    | 描述                          | 类型 | 复位  |
|------|-----------------------------|----|-----|
| 31:8 | 保留。                         | -  | -   |
| 7:4  | 版本: 读作 `ImpDef              | 只读 | 0x2 |
| 3    | <b>JEDEC</b> : 读取值为 0b1     | 只读 | 0x1 |
| 2:0  | <b>DES_0</b> : 读取值为 `ImpDef | 只读 | 0x3 |

## M33：TRCPIDR3 寄存器

偏移: 0x41fec

### 描述

TRCPIDR3

表326. TRCPIDR3  
寄存器

| 位    | 描述                           | 类型 | 复位  |
|------|------------------------------|----|-----|
| 31:8 | 保留。                          | -  | -   |
| 7:4  | <b>REVAND</b> : 读取值为 `ImpDef | 只读 | 0x0 |
| 3:0  | <b>CMOD</b> : 读取值为 `ImpDef   | 只读 | 0x0 |

## M33：TRCCIDR0 寄存器

偏移: 0x41ff0

### 描述

TRCCIDR0

表327. TRCCIDR0  
寄存器

| 位    | 描述                               | 类型 | 复位   |
|------|----------------------------------|----|------|
| 31:8 | 保留。                              | -  | -    |
| 7:0  | <b>PRMBL_0</b> : 读取值为 0b00001101 | 只读 | 0x0d |

## M33：TRCCIDR1 寄存器

偏移: 0x41ff4

### 描述

TRCCIDR1

表328. TRCCIDR1  
寄存器

| 位    | 描述                           | 类型 | 复位  |
|------|------------------------------|----|-----|
| 31:8 | 保留。                          | -  | -   |
| 7:4  | <b>CLASS</b> : 读取值为 0b1001   | 只读 | 0x9 |
| 3:0  | <b>PRMBL_1</b> : 读取值为 0b0000 | 只读 | 0x0 |

## M33：TRCCIDR2 寄存器

偏移地址: 0x41ff8

### 描述

TRCCIDR2

表 329. TRCCIDR2  
寄存器

| 位    | 描述                               | 类型 | 复位   |
|------|----------------------------------|----|------|
| 31:8 | 保留。                              | -  | -    |
| 7:0  | <b>PRMBL_2</b> : 读取值为 0b00000101 | 只读 | 0x05 |

## M33: TRCCIDR3 寄存器

偏移地址: 0x41ffc

### 描述

TRCCIDR3

表 330. TRCCIDR3  
寄存器

| 位    | 描述                               | 类型 | 复位   |
|------|----------------------------------|----|------|
| 31:8 | 保留。                              | -  | -    |
| 7:0  | <b>PRMBL_3</b> : 读取值为 0b10110001 | 只读 | 0xb1 |

## M33: CTICONTROL 寄存器

偏移地址: 0x42000

### 描述

CTI控制寄存器

表 331.  
CTICONTROL 寄存器

| 位    | 描述                       | 类型 | 复位  |
|------|--------------------------|----|-----|
| 31:1 | 保留。                      | -  | -   |
| 0    | <b>GLBEN</b> : 启用或禁用 CTI | 读写 | 0x0 |

## M33: CTIINTACK 寄存器

偏移地址: 0x42010

### 描述

CTI 中断确认寄存器

表 332. CTIINTACK  
寄存器

| 位    | 描述                                                                                                        | 类型 | 复位   |
|------|-----------------------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                                       | -  | -    |
| 7:0  | <b>INTACK</b> : 确认相应的 ctitrigout 输出。该寄存器的每个位对应一个 ctitrigout 输出。当向该寄存器的某个位写入 1 时，对应的 ctitrigout 将被确认，并被清除。 | 读写 | 0x00 |

## M33: CTIAPPSET 寄存器

偏移: 0x42014

### 描述

CTI 应用触发设置寄存器

表 333. CTIAPPSET 寄存器

| 位    | 描述                                                    | 类型 | 复位  |
|------|-------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                   | -  | -   |
| 3:0  | <b>APPSET:</b> 将某个位设置为高电平会为所选通道生成通道事件。该寄存器的每个位对应一个通道。 | 读写 | 0x0 |

## M33: CTIAPPCLEAR 寄存器

偏移: 0x42018

### 描述

CTI 应用触发清除寄存器

表 334. CTIAPPCLEAR 寄存器

| 位    | 描述                                                   | 类型 | 复位  |
|------|------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                  | -  | -   |
| 3:0  | <b>APPCLEAR:</b> 将 CTIAPPSET 中对应的位清零。该寄存器的每个位对应一个通道。 | 读写 | 0x0 |

## M33: CTIAPPPULSE 寄存器

偏移: 0x4201c

### 说明

CTI 应用脉冲寄存器

表 335. CTIAPPPULSE 寄存器

| 位    | 描述                                                      | 类型 | 复位  |
|------|---------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                     | -  | -   |
| 3:0  | <b>APPULSE:</b> 将该位设置为高电平会为所选通道生成通道事件脉冲。该寄存器的每个位对应一个通道。 | 读写 | 0x0 |

## M33: CTIINENO、CTIINEN1、...、CTIINEN6、CTIINEN7 寄存器

偏移: 0x42020、0x42024、...、0x42038、0x4203c

### 描述

CTI 触发至通道使能寄存器

表 336. CTIINENO、CTIINEN1、...、CTIINEN6、CTIINEN7 寄存器

| 位    | 描述                                                               | 类型 | 复位  |
|------|------------------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                              | -  | -   |
| 3:0  | <b>TRIGINEN:</b> 当 ctitrigin 输入被激活时，启用对应通道的交叉触发事件。该字段为每个四个通道各设一位 | 读写 | 0x0 |

## M33: CTIOUTENO、CTIOUTEN1、...、CTIOUTEN6、CTIOUTEN7 寄存器

偏移量: 0x420a0, 0x420a4, ..., 0x420b8, 0x420bc

### 描述

CTI 触发至通道使能寄存器

表 337. CTIOUTENO、CTIOUTEN1、...、CTIOUTEN6、CTIOUTEN7 寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:4 | 保留。 | -  | -  |

| 位   | 描述                                                                    | 类型 | 复位  |
|-----|-----------------------------------------------------------------------|----|-----|
| 3:0 | <b>TRIGOUTEN:</b> 使能在对应通道激活时，产生交叉触发事件传递至 ctitrigout。该字段中每个通道对应一位，共四位。 | 读写 | 0x0 |

## M33: CTITRIGINSTATUS 寄存器

偏移量: 0x42130

### 描述

CTI触发至通道使能寄存器

表 338。  
CTITRIGINSTATUS  
寄存器

| 位    | 描述                                                                                        | 类型 | 复位   |
|------|-------------------------------------------------------------------------------------------|----|------|
| 31:8 | 保留。                                                                                       | -  | -    |
| 7:0  | <b>TRIGINSTATUS:</b> 显示 ctitrigin 输入信号的状态。该字段为每个触发输入信号设一位。因寄存器显示原始 ctitrigin 输入信号，复位值为未知。 | 只读 | 0x00 |

## M33: CTITRIGOUTSTATUS 寄存器

偏移量: 0x42134

### 描述

CTI 输入触发状态寄存器

表 339。  
CTITRIGOUTSTATUS  
寄存器

| 位    | 描述                                                       | 类型 | 复位   |
|------|----------------------------------------------------------|----|------|
| 31:8 | 保留。                                                      | -  | -    |
| 7:0  | <b>TRIGOUTSTATUS:</b> 显示 ctitrigout 输出状态。字段中每个位对应一个触发输出。 | 只读 | 0x00 |

## M33: CTICHINSTATUS 寄存器

偏移: 0x42138

### 描述

CTI 通道输入状态寄存器

表 340。  
CTICHINSTATUS  
寄存器

| 位    | 描述                                                      | 类型 | 复位  |
|------|---------------------------------------------------------|----|-----|
| 31:4 | 保留。                                                     | -  | -   |
| 3:0  | <b>CTICHOUTSTATUS:</b> 显示 ctichout 输出状态。字段中每个位对应一个通道输出。 | 只读 | 0x0 |

## M33: CTIGATE 寄存器

偏移: 0x42140

### 描述

使能 CTI 通道门控寄存器

表 341。CTIGATE  
寄存器

| 位    | 描述                                           | 类型 | 复位  |
|------|----------------------------------------------|----|-----|
| 31:4 | 保留。                                          | -  | -   |
| 3    | <b>CTIGATEEN3:</b> 使能 ctichout3。置 0 以禁用通道传播。 | 读写 | 0x1 |
| 2    | <b>CTIGATEEN2:</b> 使能 ctichout2。置 0 以禁用通道传播。 | 读写 | 0x1 |

| 位 | 描述                                            | 类型 | 复位  |
|---|-----------------------------------------------|----|-----|
| 1 | <b>CTIGATEEN1</b> : 使能 ctichout1。置 0 以禁用通道传播。 | 读写 | 0x1 |
| 0 | <b>CTIGATEENO</b> : 使能 ctichout0。置 0 以禁用通道传播。 | 读写 | 0x1 |

## M33: ASICCTL 寄存器

偏移: 0x42144

### 描述

外部多路复用控制寄存器

表 342. ASICCTL 寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: ITCHOUT 寄存器

偏移: 0x42ee4

### 描述

集成测试通道输出寄存器

表 343. ITCHOUT 寄存器

| 位    | 描述                                | 类型 | 复位  |
|------|-----------------------------------|----|-----|
| 31:4 | 保留。                               | -  | -   |
| 3:0  | <b>CTCHOUT</b> : 设置 ctichout 输出的值 | 读写 | 0x0 |

## M33: ITTRIGOUT 寄存器

偏移: 0x42ee8

### 描述

集成测试触发输出寄存器

表 344. ITTRIGOUT 寄存器

| 位    | 描述                                   | 类型 | 复位   |
|------|--------------------------------------|----|------|
| 31:8 | 保留。                                  | -  | -    |
| 7:0  | <b>CTTRIGOUT</b> : 设置 cttrigout 输出的值 | 读写 | 0x00 |

## M33: ITCHIN 寄存器

偏移: 0x42ef4

### 描述

集成测试通道输入寄存器

表 345. ITCHIN 寄存器

| 位    | 描述                               | 类型 | 复位  |
|------|----------------------------------|----|-----|
| 31:4 | 保留。                              | -  | -   |
| 3:0  | <b>CTCHIN</b> : 读取 ctichin 输入的值。 | 只读 | 0x0 |

## M33: ITCTRL 寄存器

偏移: 0x42f00

### 描述

集成模式控制寄存器

表346. ITCtrl 寄存器

| 位    | 描述                  | 类型 | 复位  |
|------|---------------------|----|-----|
| 31:1 | 保留。                 | -  | -   |
| 0    | <b>IME</b> : 集成模式启用 | 读写 | 0x0 |

## M33: DEVARCH 寄存器

偏移: 0x42fbc

### 说明

设备架构寄存器

表347. DEVARCH 寄存器

| 位     | 描述                                | 类型 | 复位     |
|-------|-----------------------------------|----|--------|
| 31:21 | <b>ARCHITECT</b> : 指示组件架构         | 只读 | 0x23b  |
| 20    | <b>PRESENT</b> : 指示DEVARCH寄存器是否存在 | 只读 | 0x1    |
| 19:16 | <b>REVISION</b> : 指示架构版本号         | 只读 | 0x0    |
| 15:0  | <b>ARCHID</b> : 指示组件标识            | 只读 | 0x1a14 |

## M33: DEVID 寄存器

偏移: 0x42fc8

### 说明

设备配置寄存器

表348. DEVID 寄存器

| 位     | 描述                                                                                                     | 类型 | 复位   |
|-------|--------------------------------------------------------------------------------------------------------|----|------|
| 31:20 | 保留。                                                                                                    | -  | -    |
| 19:16 | <b>NUMCH</b> : 可用ECT通道数                                                                                | 只读 | 0x4  |
| 15:8  | <b>NUMTRIG</b> : 可用ECT触发器数                                                                             | 只读 | 0x08 |
| 7:5   | 保留。                                                                                                    | -  | -    |
| 4:0   | <b>EXTMUXNUM</b> : 指示用于asicctl的触发输入与触发输出上的多路复用器数量，默认值0b00000表示无多路复用，该位的值取决于Verilog中的EXT MUXNUM定义，需相应调整 | 只读 | 0x00 |

## M33: DEVTYPE 寄存器

偏移: 0x42fcc

### 描述

设备类型标识寄存器

表349. DEVTYPE 寄存器

| 位    | 描述                                                  | 类型 | 复位  |
|------|-----------------------------------------------------|----|-----|
| 31:8 | 保留。                                                 | -  | -   |
| 7:4  | <b>SUB</b> : 根据ARM体系结构规范，作为主分类（MAJOR字段）内调试组件类型的子分类。 | 只读 | 0x1 |
| 3:0  | <b>MAJOR</b> : 根据ARM体系结构规范，指定该调试和跟踪组件类型的主分类。        | 只读 | 0x4 |

## M33: PIDR4 寄存器

偏移: 0x42fd0

### 描述

CoreSight 外设 ID4

表 350. *PIDR4*  
寄存器

| 位    | 描述                                                            | 类型 | 复位  |
|------|---------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                           | -  | -   |
| 7:4  | 大小: 始终为0b0000。 表示该设备仅占用4KB内存                                  | 只读 | 0x0 |
| 3:0  | <b>DES_2:</b> PIDR1.DES_0、PIDR2.DES_1及PIDR4.DES_2共同标识该组件的设计者。 | 只读 | 0x4 |

## M33: PIDR5 寄存器

偏移: 0x42fd4

### 描述

CoreSight 外设 ID5

表 351. *PIDR5*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: PIDR6 寄存器

偏移: 0x42fd8

### 描述

CoreSight 外设 ID6

表 352. *PIDR6*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: PIDR7 寄存器

偏移: 0x42fdc

### 描述

CoreSight 外设 ID7

表 353. *PIDR7*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## M33: PIDR0 寄存器

偏移: 0x42fe0

### 描述

CoreSight 外设 ID0

表 354. *PIDR0*  
寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:8 | 保留。 | -  | -  |

| 位   | 描述                                             | 类型 | 复位   |
|-----|------------------------------------------------|----|------|
| 7:0 | <b>PART_0:</b> 组件12位零件编号的位[7:0]。该零件编号由组件设计者分配。 | 只读 | 0x21 |

## M33: PIDR1 寄存器

偏移: 0x42fe4

### 描述

CoreSight 外设 ID1

表 355. PIDR1  
寄存器

| 位    | 描述                                                              | 类型 | 复位  |
|------|-----------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                             | -  | -   |
| 7:4  | <b>DES_0:</b> PIDR1.DES_0、PIDR2.DES_1及PIDR4.DES_2共同标识组件设计者<br>◦ | 只读 | 0xb |
| 3:0  | <b>PART_1:</b> 组件12位零件编号的位[11:8]。该零件编号由组件设计者分配。                 | 只读 | 0xd |

## M33: PIDR2 寄存器

偏移量: 0x42fe8

### 描述

CoreSight 外设 ID2

表 356. PIDR2  
寄存器

| 位    | 描述                                                                 | 类型 | 复位  |
|------|--------------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                                | -  | -   |
| 7:4  | 版本: 本设备为 r1p0 版本                                                   | 只读 | 0x0 |
| 3    | <b>JEDEC:</b> 始终为 1。表示使用了 JEDEC 分配的设计者 ID。                         | 只读 | 0x1 |
| 2:0  | <b>DES_1:</b> PIDR1.DES_0、PIDR2.DES_1 与 PIDR4.DES_2 共同识别组件设计者<br>◦ | 只读 | 0x3 |

## M33: PIDR3 寄存器

偏移量: 0x42fec

### 描述

CoreSight 外设 ID3

表 357. PIDR3  
寄存器

| 位    | 描述                                                                                                                                            | 类型 | 复位  |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                                                                                                           | -  | -   |
| 7:4  | <b>REVAND:</b> 表示针对正在使用的组件版本所做的次要缺陷修正，例如<br>实施后的金属修正。在大多数情况下，该字段值为 0b0000。ARM 建议组件设<br>计者确保在必要时可通过寄存器（该寄存器复位值为 0b0000）驱动该字段，<br>以允许金属修正更改此字段。 | 只读 | 0x0 |
| 3:0  | <b>CMOD:</b> 客户修改。表示客户是否更改了组件的行为。在大多数情况下，该字<br>段值为 0b0000。客户在对该组件进行授权修改时应更改此值。                                                                | 只读 | 0x0 |

## M33: CIDR0 寄存器

偏移: 0x42ff0

**描述**

CoreSight 组件 ID0

表 358。*CIDR0*  
寄存器

| 位    | 描述                                          | 类型 | 复位   |
|------|---------------------------------------------|----|------|
| 31:8 | 保留。                                         | -  | -    |
| 7:0  | <b>PRMBL_0</b> : Preamble[0]。包含组件识别码的位[7:0] | 只读 | 0x0d |

**M33：CIDR1 寄存器**

偏移: 0x42ff4

**描述**

CoreSight 组件 ID1

表 359。*CIDR1*  
寄存器

| 位    | 描述                                                                    | 类型 | 复位  |
|------|-----------------------------------------------------------------------|----|-----|
| 31:8 | 保留。                                                                   | -  | -   |
| 7:4  | <b>CLASS</b> : 组件类别，例如该组件是只读存储器表或通用 CoreSight 组件。<br>包含组件识别码的位[15:12] | 只读 | 0x9 |
| 3:0  | <b>PRMBL_1</b> : Preamble[1]。包含组件识别码的位[11:8]                          | 只读 | 0x0 |

**M33：CIDR2 寄存器**

偏移: 0x42ff8

**描述**

CoreSight 组件 ID2

表 360。*CIDR2*  
寄存器

| 位    | 描述                                            | 类型 | 复位   |
|------|-----------------------------------------------|----|------|
| 31:8 | 保留。                                           | -  | -    |
| 7:0  | <b>PRMBL_2</b> : Preamble[2]。包含组件识别码的位[23:16] | 只读 | 0x05 |

**M33：CIDR3 寄存器**

偏移量: 0x42ffc

**说明**

CoreSight 组件 ID3

表 361。*CIDR3*  
寄存器

| 位    | 描述                                              | 类型 | 复位   |
|------|-------------------------------------------------|----|------|
| 31:8 | 保留。                                             | -  | -    |
| 7:0  | <b>PRMBL_3</b> : Preamble[3]。包含组件识别码的第[31:24]位。 | 只读 | 0xb1 |

**3.7.5.1. Cortex-M33 EPPB 寄存器**

EPPB（扩展私有外设总线）包含由 Raspberry Pi 实现并集成于 Cortex-M33 PPB 中的寄存器，用于为特定 RP2350 功能提供每核控制。这些寄存器每个核心拥有一份

(它们为核心本地寄存器)，且在核心热复位时被重置。

这些寄存器的基地址起始于 `0xe0080000`，在 SDK 中定义为 EPPB\_BASE。

表 362. M33\_EPPB 寄存器列表

| 偏移量 | 名称        | 说明                                            |
|-----|-----------|-----------------------------------------------|
| 0x0 | NMI_MASK0 | IRQ 0 至 31 的 NMI 掩码。该寄存器为核心本地寄存器，处理器热复位时将被重置。 |
| 0x4 | NMI_MASK1 | IRQ 0 至 51 的 NMI 掩码。该寄存器为核心本地寄存器，处理器热复位时将被重置。 |
| 0x8 | SLEEPCTRL | 非标准睡眠控制寄存器                                    |

## M33\_EPPB：NMI\_MASK0 寄存器

偏移: 0x0

表 363.  
NMI\_MASK0 寄存器

| 位    | 描述                                            | 类型 | 复位         |
|------|-----------------------------------------------|----|------------|
| 31:0 | IRQ 0 至 31 的 NMI 掩码。该寄存器为核心本地寄存器，处理器热复位时将被重置。 | 读写 | 0x00000000 |

## M33\_EPPB：NMI\_MASK1 寄存器

偏移量: 0x4

表 364.  
NMI\_MASK1 寄存器

| 位     | 描述                                            | 类型 | 复位      |
|-------|-----------------------------------------------|----|---------|
| 31:20 | 保留。                                           | -  | -       |
| 19:0  | IRQ 0 至 51 的 NMI 掩码。该寄存器为核心本地寄存器，处理器热复位时将被重置。 | 读写 | 0x00000 |

## M33\_EPPB：SLEEPCTRL 寄存器

偏移: 0x8

### 描述

非标准睡眠控制寄存器

表 365.  
SLEEPCTRL 寄存器

| 位    | 描述                                                                                                                                                                                                  | 类型 | 复位  |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:3 | 保留。                                                                                                                                                                                                 | -  | -   |
| 2    | <b>WICENACK:</b> 来自处理器中断控制器的状态信号。对 WICENREQ 的更改最终会反映在 WICENACK 上。                                                                                                                                   | 只读 | 0x0 |
| 1    | <b>WICENREQ:</b> 请求下一次处理器深度睡眠采用 WIC 睡眠模式。设置该位后，进入睡眠前应轮询 WICENACK，以确保处理器中断控制器已确认该更改。                                                                                                                 | 读写 | 0x1 |
| 0    | <b>LIGHT_SLEEP:</b> 默认情况下，任何处理器休眠都会取消系统级时钟请求的断言。重新启用时钟将在唤醒时产生5个周期的额外延迟。<br><br>将LIGHT_SLEEP设置为1可在正常休眠期间（Arm SCR.SLEEPDEEP = 0）保持时钟请求的断言，以实现更快的唤醒。处理器深度休眠（Arm SCR.SLEEPDEEP = 1）不受影响，始终取消系统级时钟请求的断言。 | 读写 | 0x0 |

## 3.8. Hazard3 处理器

Hazard3是一款低面积、高性能的RISC-V处理器，采用3级有序流水线。RP2350配置了以下标准RISC-V扩展：

- **RV32I**: 32位基础指令集
- **M**: 整数乘除及取模指令
- **A**: 原子内存操作指令
- **C**: 压缩16位指令（同样记为 **Zca**）
- **Zba**: 地址生成指令
- **Zbb**: 基本位操作指令
- **Zbs**: 单个位操作指令
- **Zbkb**: 标量密码学的基本位操作
- **Zcb**: 基本附加压缩指令
- **Zcmp**: 压缩的压栈/出栈及双重移动指令
- **Zicsr**: CSR访问指令
- 调试、机器模式及用户执行模式
- 物理内存保护单元（PMP），包含八个区域，32字节粒度，采用NAPOT
- 支持四个指令地址触发器的外部调试功能

此外，RP2350 启用了以下 Hazard3 定制扩展：

- **Xh3power**: 电源管理指令与 CSR 寄存器
- **Xh3bextm**: 位提取多重指令（用于 bootrom）
- **Xh3irq**: 具嵌套和优先级支持的本地中断控制器
- **Xh3ppmp**: 解锁的 M 模式 PMP 区域

### Hazard3 源代码

Hazard3 所有硬件源文件均在 Apache 2.0 许可协议下开放，网址：

[github.com/wren6991/hazard3](https://github.com/wren6991/hazard3)

### 3.8.1. 指令集参考

本节为 Hazard3 支持指令的程序员参考指南，涵盖基本汇编语法、指令行为、立即数范围及指令压缩条件。索引按字母顺序列出指令，包括伪指令。

本指南中的伪代码仅供参考，不能替代第3.8.1.1节中官方RISC-V规范。然而，一旦您阅读了相关规范，它应成为有效的助记工具。

#### 3.8.1.1 RISC-V规范链接

本表提供了Hazard3实现的基本指令集及其扩展的正式批准版本链接，系本参考指南所载指令的权威依据。

| 扩展                         | 规范                    |
|----------------------------|-----------------------|
| <code>RV32I</code> v2.1    | 非特权ISA 20191213       |
| <code>M</code> v2.0        | 非特权ISA 20191213       |
| <code>A</code> v2.1        | 非特权ISA 20191213       |
| <code>C</code> v2.0        | 非特权ISA 20191213       |
| <code>Zicsr</code> v2.0    | 非特权ISA 20191213       |
| <code>Zifencei</code> v2.0 | 非特权ISA 20191213       |
| <code>Zba</code> v1.0.0    | 位操作ISA扩展 20210628     |
| <code>Zbb</code> v1.0.0    | 位操作ISA扩展 20210628     |
| <code>Zbs</code> v1.0.0    | 位操作ISA扩展 20210628     |
| <code>Zbkb</code> v1.0.1   | 标量密码学ISA扩展 20220218   |
| <code>Zcb</code> v1.0.3-1  | 代码大小缩减扩展冻结版 v1.0.3-1  |
| <code>Zcmp</code> v1.0.3-1 | 代码大小缩减扩展冻结版 v1.0.3-1  |
| 机器ISA v1.12                | 特权架构 20211203         |
| 调试 v0.13.2                 | RISC-V外部调试支持 20190322 |

您也可以参考RISC-V汇编程序员手册以了解汇编语法信息。

有关实现定义行为的详细问题，请查阅源码，该部分不包含在RISC-V规范中。RP2350使用版本86fc4e3，针对提交2f6e983和af08c0b进行了金属ECO修订。

### 3.8.1.2 架构字符串

`-march`字符串完整指定了可用的RISC-V指令集，编译器可据此为您的设备生成正确且最优代码。请按以下优先顺序依次使用：

1. 使用`rv32ima_zicsr_zifencei_zba_zbb_zbs_zbkb_zca_zcb_zcmp`，适用于支持`Zcb`和`Zcmp`扩展的编译器，如GCC 14。
2. 使用`rv32ima_zicsr_zifencei_zba_zbb_zbs_zbkb_zca_zcb`，适用于配备较旧汇编器且不支持`Zcmp`的GCC 14。
3. 使用`rv32imac_zicsr_zifencei_zba_zbb_zbs_zbkb`，适用于较旧的编译器，如GCC 13及更早版本。

### 3.8.1.3. RISC-V 架构状态

程序员可见的可变状态包括：

- 31个32位整数通用寄存器(GPR)，命名为`x1`至`x31`
- 程序计数器`pc`，指向内存中当前指令的起始位置
- 控制和状态寄存器(CSR)，用于配置处理器行为并用于陷阱处理
- 本地监视位，有助于保持原子读-改-写序列的正确性
- 当前特权级，决定内核可访问的内存地址、可访问的CSR以及可执行的指令

Hazard3支持两个特权级：机器级和用户级。两者可交替称为模式，通常简称M模式和U模式。调试模式作为高于M模式的额外特权级存在。

第0号通用寄存器 `x0` 被硬连线为零，写入操作被忽略。

无标志寄存器；分支指令直接执行通用寄存器之间的比较。

该状态在每个硬件线程（`hart`）中复制。RP2350实现了两个Hazard3核心，每个核心含一个`hart`。

### 3.8.1.3.1. 寄存器约定

以下ABI名称与 `x0` 至 `x31` 同义：

| 寄存器                    | ABI名称                 | 描述           |
|------------------------|-----------------------|--------------|
| <code>x0</code>        | <code>zero</code>     | 硬连线为零；写入时被忽略 |
| <code>x1</code>        | <code>ra</code>       | 返回地址（链接寄存器）  |
| <code>x2</code>        | <code>sp</code>       | 栈指针          |
| <code>x3</code>        | <code>gp</code>       | 全局指针         |
| <code>x4</code>        | <code>tp</code>       | 线程指针         |
| <code>x5 - x7</code>   | <code>t0 - t2</code>  | 临时寄存器        |
| <code>x8</code>        | <code>s0 或 fp</code>  | 保存寄存器或帧指针    |
| <code>x9</code>        | <code>s1</code>       | 已保存寄存器       |
| <code>x10 - x11</code> | <code>a0 - a1</code>  | 函数参数和返回值     |
| <code>x12 - x17</code> | <code>a2 - a7</code>  | 函数参数         |
| <code>x18 - x27</code> | <code>s2 - s11</code> | 已保存寄存器       |
| <code>x28 - x31</code> | <code>t3 - t6</code>  | 临时寄存器        |

寄存器 `x1` 至 `x31` 均相同，任何32位操作码均可使用这些寄存器的任意组合。然而，压缩指令对常用寄存器 `sp`、`ra`、`s0`、`s1` 及 `a0` 至 `a5` 给予优先照顾，以提升代码密度。Hazard3实现的所有压缩指令都是现有32位指令的16位别名，因此仍可对任何寄存器执行任何操作。

有关ABI寄存器分配及RISC-V过程调用约定的详细信息，请参阅RISC-V PSABI规范。

### 3.8.1.4. 压缩指令

Hazard3 实现的 RISC-V 扩展使用了 32 位和 16 位操作码的混合，其中后者称为**压缩指令**。除 `Zcmp` 外，每条压缩指令均映射至现有 32 位指令的一个子集。例如，`c.add` 是 `add` 指令的 16 位别名，但对寄存器分配有特定限制。

汇编器在可能的情况下会自动使用压缩指令。例如，`add a0, a0, a1` 是可压缩形式的 `add` 指令。启用汇编器中的压缩指令时，该指令会被组装为 16 位操作码 `c.add a0, a1`。

以下扩展使用 16 位操作码：

- **C：压缩指令（非浮点子集等同表示为 Zca）**
- **Zcb：附加基础压缩指令**
- **Zcmp：压缩的入栈、出栈及双寄存器移动指令**

禁用上述扩展进行编译（及汇编）时，所有指令均对齐至 32 位边界。对于分支密集型代码序列（参见第3.8.7节），这可能带来轻微的性能优势，但代价是代码密度降低。

当指令具有可选的16位压缩形式时，压缩形式的限制会在32位形式的清单中作详细说明。在针对代码大小进行优化时，了解这些限制非常重要。若未提及此类限制，则表示该指令始终为32位操作码。

`Zcmp`是一个例外，其指令各自扩展为一系列来自RV32I基础指令集的32位指令，因此它们没有直接对应的32位指令。

### 3.8.1.5. 伪代码约定

本节的伪代码采用Verilog 2005语法（IEEE 1364-2005）。以下Verilog运算符贯穿全文使用：

- 中缀运算符`+`、`-`、`*`、`/`、`&`、`^`、`|`、`<<`、`==`、`!=`、`<及>`可视为与对应的C语言运算符相同。
- `$signed()`位转换为有符号值；两个有符号值之间的比较为有符号比较。
- `>>`始终表示逻辑（零扩展）右移。
- `>>>`对有符号值表示算术（符号扩展）右移。
- `{a, b}`是`a`和`b`的位连接，`a`位于结果的更高有效位。
- `a[n]`对数组的访问是下标访问。例如`mem[0]`表示内存的第一个字节。
- `x[m:l]`在打包数组（位向量）上是位切片，其中`m`为（包含）最高有效位，`l`为（包含）最低有效位。例如`rs1[7:0]`是`rs1`的最低8位。
- `{n{x}}`其中`n`为常数，`x`为打包数组，表示将`x`重复`n`次。`n`个`x`被连接在一起。例如`{32{1'b1}}`是一个32位的全一值。

伪代码使用`<=`非阻塞赋值来分配输出：所有此类赋值在伪代码块执行完成后统一应用。局部变量可以使用`=`阻塞赋值，立即更新被赋值对象，类似于如C语言程序中的`=`过程式赋值。该区别在某些情况下尤为重要，例如`rd`与`rs1`可能别名同一寄存器，但一般只需注意`a <= b`与`a = b`均为对`a`的赋值即可。

#### 3.8.1.5.1. 伪代码中使用的变量

本指南中的伪代码使用以下变量约定：

- `rs1`、`rs2`及`rd`为32位无符号打包数组（位向量），代表两个寄存器操作数及目标寄存器的值。
- `regnum_rs1`、`regnum_rs2`与`regnum_rd`是选择对应于`rs1`、`rs2`和`rd`通用寄存器的5位寄存器编号。
- `imm`是一个32位无符号打包数组，表示指令的立即数值。
- `pc`是一个32位无符号打包数组，表示程序计数器，即当前指令的精确地址。
- `mem`是一个8位无符号打包数组的数组，每个元素对应内存中的一个字节地址。
- `csr`是一个32位无符号打包数组的数组，每个元素对应第3.8.9节中列出的CSR。
- `priv`是一个2位无符号打包数组，当内核处于调试（Debug）或机器模式（M-mode）时，其值为`0x3`；处于用户模式（U-mode）时，其值为`0x0`。
- `i`和`j`为伪代码中的临时变量，类型为整数（integer），可用作循环变量。

以下任务在整个过程中广泛使用：

- `raise_exception(n)`引发一个原因是`n`的异常（参见第3.8.4.1节）。
- `bus_error(addr)`当地址`addr`返回总线错误时返回`1`，否则返回`0`。

### 3.8.1.6. 指令字母顺序列表

本指令参考涵盖Hazard3在RP2350上实现的所有扩展指令。下表还包括常见伪指令，如 `not` 和 `ret`，您可能在反汇编中见到，但在ISA手册中未见收录。伪指令的链接指向其对应的底层硬件指令条目。

#### ● 提示

指令列表左侧边缘的指令名称为返回本索引的链接。使用这些链接可快速返回此处并查找其他指令。

| 字母顺序：先从左到右，后从上到下。        |                           |                           |                            |                          |                           |
|--------------------------|---------------------------|---------------------------|----------------------------|--------------------------|---------------------------|
| 加                        | <a href="#">addi</a>      | <a href="#">amoadd.w</a>  | <a href="#">amoand.w</a>   | <a href="#">amomax.w</a> | <a href="#">amomaxu.w</a> |
| <a href="#">amomin.w</a> | <a href="#">amominu.w</a> | <a href="#">amoor.w</a>   | <a href="#">amoswap.w</a>  | <a href="#">amoxor.w</a> | <a href="#">and</a>       |
| <a href="#">andi</a>     | <a href="#">andn</a>      | <a href="#">auipc</a>     | <a href="#">bclr</a>       | <a href="#">bclri</a>    | <a href="#">beq</a>       |
| <a href="#">beqz</a>     | <a href="#">bext</a>      | <a href="#">bexti</a>     | <a href="#">bge</a>        | <a href="#">bgeu</a>     | <a href="#">bgez</a>      |
| <a href="#">bgt</a>      | <a href="#">bgtu</a>      | <a href="#">bgtz</a>      | <a href="#">binv</a>       | <a href="#">binvi</a>    | <a href="#">ble</a>       |
| <a href="#">bleu</a>     | <a href="#">blez</a>      | <a href="#">blt</a>       | <a href="#">bltu</a>       | <a href="#">bltz</a>     | <a href="#">bne</a>       |
| <a href="#">bnez</a>     | <a href="#">brev8</a>     | <a href="#">bset</a>      | <a href="#">bseti</a>      | <a href="#">clz</a>      | <a href="#">cm.mva01s</a> |
| <a href="#">cm.mvs01</a> | <a href="#">cm.pop</a>    | <a href="#">cm.popret</a> | <a href="#">cm.popretz</a> | <a href="#">cm.push</a>  | <a href="#">cpop</a>      |
| <a href="#">csrc</a>     | <a href="#">csrci</a>     | <a href="#">csrr</a>      | <a href="#">csrrc</a>      | <a href="#">csrrci</a>   | <a href="#">csrrs</a>     |
| <a href="#">csrrsi</a>   | <a href="#">csrrw</a>     | <a href="#">csrrwi</a>    | <a href="#">csrs</a>       | <a href="#">csrsi</a>    | <a href="#">csrsw</a>     |
| <a href="#">csrwi</a>    | <a href="#">ctz</a>       | <a href="#">div</a>       | <a href="#">divu</a>       | <a href="#">ebreak</a>   | <a href="#">ecall</a>     |
| <a href="#">fence</a>    | <a href="#">fence.i</a>   | <a href="#">j</a>         | <a href="#">jal</a>        | <a href="#">jalr</a>     | <a href="#">jr</a>        |
| <a href="#">lb</a>       | <a href="#">lbu</a>       | <a href="#">lh</a>        | <a href="#">lhu</a>        | <a href="#">lr.w</a>     | <a href="#">lui</a>       |
| <a href="#">lw</a>       | <a href="#">max</a>       | <a href="#">maxu</a>      | <a href="#">min</a>        | <a href="#">minu</a>     | <a href="#">mret</a>      |
| <a href="#">mul</a>      | <a href="#">mulh</a>      | <a href="#">mulhsu</a>    | <a href="#">mulhu</a>      | <a href="#">mv</a>       | <a href="#">neg</a>       |
| <a href="#">nop</a>      | <a href="#">not</a>       | <a href="#">or</a>        | <a href="#">orc.b</a>      | <a href="#">ori</a>      | <a href="#">orn</a>       |
| <a href="#">pack</a>     | 打包高                       | 余数                        | 重新移动                       | 返回                       | <a href="#">rev8</a>      |
| 循环左移                     | 循环右移                      | 循环右移带进位                   | 设置位                        | <a href="#">sc.w</a>     | 等于零顺序                     |
| 符号扩展字节                   | 符号扩展半字                    | 大于零设置                     | 带1加法                       | 带2加法                     | 带3加法                      |
| 带符号右移                    | <a href="#">sll</a>       | <a href="#">slli</a>      | <a href="#">slt</a>        | <a href="#">slti</a>     | <a href="#">sltiu</a>     |
| <a href="#">slt</a>      | <a href="#">slt</a>       | <a href="#">snez</a>      | <a href="#">sra</a>        | <a href="#">srai</a>     | <a href="#">srl</a>       |
| <a href="#">srl</a>      | 减                         | <a href="#">sw</a>        | 解压                         | <a href="#">wfi</a>      | <a href="#">xnor</a>      |
| <a href="#">xor</a>      | <a href="#">xori</a>      | <a href="#">zext.b</a>    | <a href="#">zext.h</a>     | <a href="#">zip</a>      |                           |

本参考指南其余部分根据扩展对指令进行分类：

- [RV32I：基础指令集架构（寄存器对寄存器）](#)
- [RV32I：基础指令集架构（寄存器对立即数）](#)
- [RV32I：基础指令集架构（大立即数）](#)
- [RV32I：基础指令集架构（控制转移）](#)
- [RV32I：基础指令集架构（加载/存储）](#)

- M：乘法与除法
- A：原子操作
- C：压缩指令
- Zba：用于地址生成的位操作扩展
- Zbb：基础位操作扩展
- Zbs：单个位操作扩展
- Zbkb：面向标量密码学的基础位操作扩展
- Zcb：附加基础压缩指令
- Zcmp：压缩的入栈、出栈及双寄存器移动指令
- RV32I 和 Zifencei：内存排序
- Zicsr：控制与状态寄存器访问
- 特权指令

### 3.8.1.7. RV32I：基础指令集架构（寄存器对寄存器）

这些指令计算两个寄存器操作数，`rs1`和`rs2`的函数值。并将32位结果写入目的寄存器`rd`。

#### 加

寄存器相加。

用法：

```
add rd, rs1, rs2
```

操作：

```
rd <= rs1 + rs2;
```

可压缩条件如下：

- 当`rd`等于`rs1`，且所有操作数均非零（即`c.add`）
- 当`rs2`为零，且`rd`和`rs1`均非零（即`c.mv`）

#### and

寄存器按位与操作。

用法：

```
and rd, rs1, rs2
```

操作：

```
rd <= rs1 & rs2;
```

可压缩条件： **rd** 等于 **rs1**，且寄存器处于 x8至 x15范围内。

**or**

寄存器按位或操作。

用法：

```
or rd, rs1, rs2
```

操作：

```
rd <= rs1 | rs2;
```

可压缩条件： **rd** 等于 **rs1**，且寄存器处于 x8至 x15范围内。

**sll**

逻辑左移。移位量取模32。

用法：

```
sll rd, rs1, rs2
```

操作：

```
rd <= rs1 << rs2[4:0];
```

**slt**

若小于（有符号）则置位。结果为 0 表示假， 1 表示真。

用法：

```
slt rd, rs1, rs2
sltz rd, rs1 // 伪指令：rs2为零
sgtz rd, rs2 // 伪指令：rs1为零
```

操作：

```
rd <= $signed(rs1) < $signed(rs2);
```

**sltu**

若小于（无符号）则置位。结果为 0 表示假， 1 表示真。

用法：

```
sltu rd, rs1, rs
snez rd, rs2 // 伪指令: rs1为零
```

操作:

```
rd <= rs1 < rs2;
```

#### sra

算术右移。移位量取模32。

用法:

```
sra rd, rs1, rs2
```

操作:

```
rd <= $signed(rs1) >>> rs2[4:0];
```

#### srl

逻辑右移。移位数取模32。

用法:

```
srl rd, rs1, rs2
```

操作:

```
rd <= rs1 >> rs2[4:0];
```

#### 减

寄存器间的二补数减法运算。

用法:

```
sub rd, rs1, rs2
neg rd, rs2 // 伪代码: rs1为零
```

操作:

```
rd <= rs1 - rs2;
```

可压缩条件: **rd**等于**rs1**, 且寄存器处于**x8**至**x15**范围内。

**xor**

寄存器与寄存器的按位异或操作

用法：

```
xor rd, rs1, rs2
```

操作：

```
rd <= rs1 ^ rs2;
```

可压缩条件： **rd** 等于 **rs1**，且寄存器处于 **x8**至**x15**范围内。

### 3.8.1.8. RV32I：基础指令集（寄存器-立即数类型）

这些指令计算一个寄存器 **rs1**与一个立即数操作数 **imm**的函数。它们将32位结果写入目标寄存器 **rd**。

立即数操作数是直接编码于指令中的常量，避免了先将常量载入寄存器的开销。

**addi**

寄存器加立即数。

用法：

```
addi rd, rs1, imm
mv rd, rs1 // 伪代码: imm 为 0
nop // 伪代码: rd、rs1 为零, imm 为 0
```

操作：

```
rd <= rs1 + imm
```

立即数范围： **-0x800**至**0x7ff**（32位），16位则更小。

可压缩条件：

- **rd** 等于 **rs1**，且立即数范围为 **-0x20**至**0x1f**（即 **c.addi**）
- **rd**不为零，**rs1**为零，且立即数范围为-0x20至**0x1f**（即**c.li**）
- **rd**在 **x8**至 **x15**之间，**rs1**为 **sp**，且立即数为非零的四的倍数，范围为0x000至0x3fc（即**c.addi4spn**）

- **rd**为 **sp**，**rs1**为 **sp**，且立即数为非零的16的倍数，范围为-0x200至**0x1f0**（即**c.addi16sp**）

注意：压缩指令 **c.mv**在规范中扩展为add，而非 **addi**。

**andi**

按位与立即数寄存器操作。

用法：

```
andi rd, rs1, imm
zext.b rd, rs1 // 伪指令: imm为0xff
```

操作:

```
rd <= rs1 & imm;
```

立即数范围: 32位为-0x800至0x7ff, 16位为-0x20至0x1f。

可压缩条件: rd与rs1相同, 寄存器位于x8至x15, 且立即数在-0x20至0x1f范围内。

**ori**

按位或立即数寄存器操作。

用法:

```
ori rd, rs1, imm
```

操作:

```
rd <= rs1 | imm;
```

立即数范围: -0x800至0x7ff

**slli**

立即数逻辑左移。

用法:

```
slli rd, rs1, imm
```

操作:

```
rd <= rs1 << imm;
```

立即数范围: 0至31。

可压缩条件: rd与rs1匹配, 且寄存器不为零。

**slti**

如果小于立即数(有符号), 则置位。结果为0表示假, 1表示真。

用法:

```
slti rd, rs1, imm
```

操作:

```
rd <= $signed(rs1) < $signed(imm);
```

立即数范围： -0x800 至 0x7ff

#### **sltiu**

如果小于立即数（无符号），则置位。结果为 0 表示假， 1 表示真。

用法：

```
sltiu rd, rs1, imm
seqz rd, rs1 // 伪指令：imm 值为 1
```

操作：

```
rd <= rs1 < imm;
```

立即数范围： -0x800 至 0x7ff

注意立即数范围中标示的负值为二补数：该指令在无符号上下文中使用它们，因此 -0x800 至 -0x001 可视为 +0xfffffff800 至 +0xffffffff 用于比较。

#### **srai**

立即数算术右移。

用法：

```
srai rd, rs1, imm
```

操作：

```
rd <= $signed(rs1) >>> imm;
```

立即数范围： 0 至 31。

仅当满足以下条件时可压缩： rd 与 rs1 相同，且寄存器位于 x8 至 x15 范围内。

#### **srlti**

逻辑右移，立即数格式。

用法：

```
srlti rd, rs1, imm
```

操作：

```
rd <= rs1 >> imm;
```

立即数范围： 0 至 31。

仅当满足以下条件时可压缩： **rd**与 **rs1**相同，且寄存器位于 **x8**至 **x15**范围内。

**xori**

寄存器与立即数按位异或。

用法：

```
xori rd, rs1, imm
not rd, rs1 // 伪指令：imm 为 -1
```

操作：

```
rd <= rs1 ^ imm;
```

立即数范围： -0x800 至 0x7ff

仅当满足以下条件时可压缩： **rd**与 **rs1**相同，寄存器位于 **x8**至 **x15**范围，且立即数为 -1（亦称 **c.not**）

**3.8.1.9. RV32I：基础指令集（大立即数）**

这些指令为两条指令序列中的第一条，用于生成32位常数或基于 **pc**的32位偏移。

**auipc**

将高位立即数加至程序计数器。

用法：

```
auipc rd, imm
```

操作：

```
rd <= pc + (imm << 12);
```

立即数范围： -0x80000 至 0x7FFFF。

注意 -0x80000 至 -0x00001 等同于 0x80000 至 0xfffff（仅限**RV32**的左移操作后），汇编器亦可能接受这些正数值。

**lui**

加载高位立即数。

用法：

```
lui rd, imm
```

操作：

```
rd <= imm << 12;
```

立即数范围：如果为32位，则为 `-0x80000` 至 `0x7fffff`；如果为16位，则为 `-0x20` 至 `0x1f`。

可压缩条件为：`rd`既非 `zero` 也非 `sp`，且 `imm` 为范围-0x20至0x1f内的非零值。

注意 `-0x80000` 至 `-0x00001` 等同于 `0x80000` 至 `0xfffff`（仅限RV32的左移操作后），汇编器亦可能接受这些正数值。

### 3.8.1.10. RV32I：基础指令集（控制流转移）

这些指令会修改 `pc` 的值。未被修改时，`pc` 按当前指令的字节数递增。

条件分支根据两个寄存器的比较结果，选择性地修改或保持 `pc` 不变。没有标志寄存器，但您可以通过将寄存器与零寄存器比较，将布尔条件传递至分支。

**beq**

相等时分支。

用法：

```
beq rs1, rs2, label
beqz rs1, label // 伪指令：rs2 为零
```

操作：

```
if (rs1 == rs2)
 pc <= label;
```

立即数范围：对32位为 `-0x1000` 至 `0x0ffe` 之间的偶数 ( $\pm 4$  kB)；对16位为 `-0x100` 至 `0x0fe` 之间 ( $\pm 256$  B)。

可压缩条件：`rs2` 为零，且立即数位于 `-0x100` 至 `0x0fe` 范围内（即 `c.beqz` 指令）。

**bge**

有符号大于等于时分支。

用法：

```
bge rs1, rs2, label
bgez rs1, label // 伪指令：rs2 为零
ble rs2, rs1, label // 伪指令：操作数由汇编器交换
blez rs2, label // 伪指令：rs1 为零
```

操作：

```
if ($signed(rs1) >= $signed(rs2))
 pc <= label;
```

立即数范围：偶数值范围为 `-0x1000` 至 `0x0ffe` ( $\pm 4$  kB)

**bgeu**

无符号小于等于条件分支。

用法：

```
bgeu rs1, rs2, label
bleu rs2, rs1, label // 伪指令：操作数由汇编器交换
```

操作：

```
if (rs1 >= rs2)
 pc <= label;
```

立即数范围：偶数值范围为 -0x1000 至 0x0ffe (±4 kB)

**blt**

有符号小于条件分支。

用法：

```
blt rs1, rs2, label
bltz rs1, label // 伪指令：rs2 为零
bgt rs2, rs1, label // 伪指令：操作数由汇编器交换
bgtz rs2, label // 伪代码：rs1 为零
```

操作：

```
if ($signed(rs1) < $signed(rs2))
 pc <= label;
```

立即数范围：偶数值范围为 -0x1000 至 0x0ffe (±4 kB)

**bltu**

无符号小于条件分支。

用法：

```
bltu rs1, rs2, label
bgtu rs2, rs1, label // 伪指令：操作数由汇编器交换
```

操作：

```
if (rs1 < rs2)
 pc <= label;
```

立即数范围：偶数值范围为 -0x1000 至 0x0ffe (±4 kB)

**bne**

不等则分支。

用法：

```
bne rs1, rs2, label
bnez rs1, label // 伪指令：rs2 为零
```

操作：

```
if (rs1 != rs2)
 pc <= label;
```

立即数范围：对32位为-0x1000至0x0ffe之间的偶数（±4 kB）；对16位为-0x100至0x0fe之间（±256 B）。

当且仅当 **rs2** 为零，且立即数位于-0x100至0x0fe之间时可压缩（即c.bnez）。

**jal**

跳转并链接，相对于 pc。

用法：

```
jal rd, label
jal label // 伪指令：rd 为 ra
j label // 伪指令：rd 为零
```

操作：

```
rd <= pc + 4; // 如果操作码为16位，则为 +2
pc <= label;
```

立即数范围：32位时，为 -0x100000 至 0x0ffffe 范围内的偶数值（±1 MB）；16位时，为 -0x800 至 0x7fe 范围内的偶数值（±2 kB）。

可压缩条件： **rd** 为零或 **ra**，且立即数范围为-0x800至0x7fe。

**jalr**

跳转并链接，寄存器偏移。

用法：

```
jalr rd, rs1, imm // (若省略, imm隐式为0。)
jalr rd, imm(rs1) // 替代语法。 (若省略, imm隐式为0。)
jalr rs1, imm // 伪指令：rd为ra。 (若省略, imm隐式为0。)
jalr imm(rs1) // 伪指令：rd为ra。 (若省略, imm隐式为0。)
jr rs1, imm // 伪指令：rd为零。 (若省略, imm隐式为0。)
jr imm(rs1) // 伪指令：rd为零。 (若省略, imm隐式为0。)
返回 // jr ra的伪指令
```

操作：

```
rd <= pc + 4; // 若指令码为16位，则为+2, pc <=
rs1 + imm;
```

立即数范围： -0x800 至 0x7ff。

可压缩条件： rd 为零或 ra，立即数为零，且 rs1 不为零。

### 3.8.1.11. RV32I：基础ISA（加载与存储）

这些指令在内存与核心寄存器之间传输数据。寄存器操作数 rs1 与立即数 imm 相加以生成地址。存储指令将寄存器操作数 rs2 写入内存，加载指令则从内存读取到目标寄存器 rd。

所有针对 RISC-V 自然对齐地址的加载和存储指令均为 **单复制原子操作**。这意味着，自然对齐的加载指令不会观察到该地址在任何自然对齐存储前后出现的字节断裂。等效地，单个自然对齐的加载或存储指令所覆盖的所有字节，均通过与内存子系统的一次事务传输完成。

Hazard3 在对非自然对齐地址执行加载或存储时触发异常。详见第 3.8.4.1 节，获取异常原因的全面列表。

#### lb

从内存加载带符号字节。

用法：

```
lb rd, imm(rs1)
lb rd, (rs1) // 若省略, imm默认为0。
```

操作：

```
reg [31:0] addr;
addr = rs1 + imm;
if (bus_fault(addr)) begin
 raise_exception(4'h5); // 原因 = 加载故障
end else begin
 rd <= {
 {24{mem[addr][7]}}, // 符号扩展
 mem[addr]
 };
end
```

立即数范围： -0x800 至 0x7ff (32位)，或 0x0 至 0x3 (16位)。

#### lbu

从内存加载无符号字节。

用法：

```
lbu rd, imm(rs1)
lbu rd, (rs1) // 若省略, imm默认为0。
```

操作：

```

reg [31:0] addr;
addr = rs1 + imm;
if (bus_fault(addr)) begin
 raise_exception(4'h5); // 原因 = 加载故障
end else begin
 rd <= {
 24'h000000,
 mem[addr]
 };
end

```

立即数范围： -0x800 至 0x7ff （32位），或 0x0 至 0x3 （16位）。

可压缩条件：rd和rs1均为x8至x15，且立即数范围为0x0至0x3。

#### lh

从内存加载有符号半字。

用法：

```

lh rd, imm(rs1)
lh rd, (rs1) // 若省略, imm 隐含为0。

```

操作：

```

reg [31:0] addr;
addr = rs1 + imm;
if (addr[0]) begin
 raise_exception(4'h4); // 原因 = 非对齐加载
end else if (bus_fault(addr)) begin
 raise_exception(4'h5); // 原因 = 读取故障
end else begin
 rd <= {
 {16{mem[addr + 1][7]}},
 mem[addr + 1],
 mem[addr]
 };
end

```

立即数范围：32位为 -0x800 至 0x7ff，或16位偶数值为 0x0 至 0x2。

满足以下情况则可压缩：rd和rs1位于x8至x15，且立即数为0x0或0x2。

#### lhu

从内存加载无符号半字。

用法：

```

lhu rd, imm(rs1)
lhu rd, (rs1) // 若省略, imm 默认为0。

```

操作：

```

reg [31:0] addr;
addr = rs1 + imm;
if (addr[0]) begin
 raise_exception(4'h4); // 原因 = 非对齐加载
end else if (bus_fault(addr)) begin
 raise_exception(4'h5); // 原因 = 读取故障
end else begin
 rd <= {
 16'h0000, // 零扩展
 mem[addr + 1],
 mem[addr]
 };
end

```

立即数范围：32位为 -0x800 至 0x7ff，或16位偶数值为 0x0 至 0x2。

满足以下情况则可压缩：rd和rs1位于x8至x15，且立即数为0x0或0x2。

#### lw

从内存加载字。

用法：

```

lw rd, imm(rs1)
lw rd, (rs1) // 若省略, imm 默认为 0。

```

操作：

```

reg [31:0] addr;
addr = rs1 + imm;
if (addr[1:0]) begin
 raise_exception(4'h4); // 原因 = 非对齐加载
end else if (bus_fault(addr)) begin
 raise_exception(4'h5); // 原因 = 读取故障
end else begin
 rd <= {
 mem[addr + 3], // 注意小端序;
 mem[addr + 2], // 最高有效位存于最高地址
 mem[addr + 1],
 mem[addr]
 };
end

```

立即数范围：-0x800 至 0x7ff（32位），16位则更小。

可压缩条件：

- rd 和 rs1 均处于 x8 至 x15，立即数为 0x00 至 0x7c 范围内的四的倍数（即 c.lw）
- rd 不为零，rs1 为 sp，且立即数为 0x00 至 0xfc 范围内的四的倍数（即 c.lwsp）

#### 设置位

将字节存储至内存。

用法：

```
sb rs2, imm(rs1)
sb rs2, (rs1) // 若省略, imm隐式为0。
```

操作：

```
reg [31:0] addr;
addr = rs1 + imm;
if (bus_fault(addr)) begin
 raise_exception(4'h7); // 原因 = 存储/AMO故障
end else begin
 mem[addr] <= rs2[7:0];
end
```

立即数范围： -0x800 至 0x7ff （32位），或 0x0 至 0x3 （16位）。

可压缩条件： rd和rs1均为x8至x15，且立即数范围为0x0至0x3。

#### 带符号右移

将半字存储至内存。

用法：

```
sh rs2, imm(rs1)
sh rs2, (rs1) // 若省略, imm隐式为0。
```

操作：

```
reg [31:0] addr;
addr = rs1 + imm;
if (addr[0]) begin
 raise_exception(4'h6); // 原因 = 非对齐存储/AMO
end else if (bus_fault(addr)) begin
 raise_exception(4'h7); // 原因 = 存储/AMO故障
end else begin
 mem[addr] <= rs2[7:0];
 mem[addr + 1] <= rs2[15:8];
end
```

立即数范围：32位为 -0x800 至 0x7ff，或 16位偶数值为 0x0 至 0x2。

满足以下情况则可压缩： rd和 rs1位于 x8至 x15，且立即数为 0x0 或 0x2。

#### SW

将字存储至内存。

用法：

```
sw rs2, imm(rs1)
sw rs2, (rs1) // 若省略, imm隐式为0。
```

操作：

```

reg [31:0] addr;
addr = rs1 + imm;
if (addr[1:0]) begin
 raise_exception(4'h6); // 原因 = 非对齐存储/AMO
end else if (bus_fault(addr)) begin
 raise_exception(4'h7); // 原因 = 存储/AMO故障
end else begin
 mem[addr] <= rs2[7:0];
 mem[addr + 1] <= rs2[15:8];
 mem[addr + 2] <= rs2[23:16];
 mem[addr + 3] <= rs2[31:24];
end

```

立即数范围：-0x800至0x7ff（32位），16位则更小。

可压缩条件：

- **rs1** 和 **rs2**位于 x8至 x15，立即数为四的倍数，范围为 0x00至 0x7c（即 **c.sw**）
- **rs2**不为零，**rs1**为 sp，且立即数为四的倍数，范围为0x00至0xfc（即**c.swsp**）

### 3.8.1.12. M：乘法和除法

这些指令实现整数乘法、除法和取模运算。

**div**

除法（带符号）。

用法：

```
div rd, rs1, rs2
```

操作：

```

if (rs2 == 32'h0)
 rd <= 32'hffffffff; // 针对除零的定义
else if (rs1 == 32'h80000000 && rs2 == 32'hffffffff)
 rd <= 32'h80000000; // 定义有符号溢出
else
 rd <= $signed(rs1) / $signed(rs2); // rd的符号为符号的异或值

```

**divu**

无符号除法。

用法：

```
divu rd, rs1, rs2
```

操作：

```

if (rs2 == 32'h0)
 rd <= 32'hffffffff; // 针对除零的定义
else
 rd <= rs1 / rs2;

```

**mul**

乘法  $32 \times 32 \rightarrow 32$ 。

用法：

```
mul rd, rs1, rs2
```

操作：

```
rd <= rs1 * rs2;
```

仅当满足以下条件时可压缩： **rd** 与 **rs1** 相同，且寄存器位于 **x8** 至 **x15** 范围内。

**mulh**

有符号（32位）乘以有符号（32位），返回64位结果的高32位。

用法：

```
mulh rd, rs1, rs2
```

操作：

```

// 两个操作数均符号扩展至64位：
reg [63:0] result_full;
result_full = {{32{rs1[31]}}, rs1} * {{32{rs2[31]}}, rs2};
rd <= result_full[63:32];

```

**mulhsu**

有符号（32位）乘以无符号（32位），返回64位结果的高32位。

用法：

```
mulhsu rd, rs1, rs2
```

操作：

```

// rs1 为符号扩展，rs2 为零扩展：
reg [63:0] result_full;
result_full = {{32{rs1[31]}}, rs1} * {32'h00000000, rs2};
rd <= result_full[63:32];

```

**mulhu**

无符号乘法（32 位乘以 32 位），返回 64 位结果的高 32 位。

用法：

```
mulhu rd, rs1, rs2
```

操作：

```
// 两个操作数均零扩展至 64 位：
reg [63:0] result_full;
result_full = {32'h00000000, rs1} * {32'h00000000, rs2};
rd <= result_full[63:32];
```

**余数**

取余（带符号）。

用法：

```
rem rd, rs1, rs2
```

操作：

```
if (rs2 == 32'h0)
 rd <= rs1; // 针对除零的定义
else
 rd <= $signed(rs1) % $signed(rs2); // rd 的符号与 rs1 相同
```

**重新移动**

取余（无符号）。

用法：

```
remu rd, rs1, rs2
```

操作：

```
if (rs2 == 32'h0)
 rd <= rs1;
else
 rd <= rs1 % rs2;
```

**3.8.1.13. A: Atomics**

这些指令帮助软件安全且并发地修改共享变量。它们分为两类：

- **lr.w**和**sc.w**，即保留加载和条件存储指令，允许软件通过循环直到成功，安全地对共享变量执行读-改-写操作。
- **amo\*.w**指令（原子内存操作，简称AMO），原子性地修改内存位置并返回修改前该位置的值。

本节伪代码引用了一个位全局变量`local_monitor_valid`。当hart满足以下条件时，该变量为真：

- 先前已成功完成一次AHB5独占读取；
- 自该读取以来未尝试过独占写入；
- 自该读取以来未被中断或未发生异常（实现定义的行为）。

伪代码在`local_monitor_valid`标志上维护该不变量。该标志帮助hart在自身中断范围内维持读-改-写序列的原子性。当本地监视器标志未设置时，硬件拒绝执行独占写入。

即使在读取阶段中止时，AMO仍会清除本地监视器状态，因为即使在此情况下，您已尝试执行包含独占写入的指令。在包含介于lr.w和sc.w之间执行AMO的lr.w、sc.w序列中，sc.w始终失败。

Hazard3基于AHB5独占访问构建其原子共享内存实现。以下任务贯穿本节，用于表示AHB5 32位独占读写操作：

```
// 从内存读取32位，并根据 // 全局监视器返回预留成功或失败。如果预留成功，则设置本地监视器位
。
task exclusive_read_32;
 input [31:0] addr;
 output [31:0] data;
 output exclusive_ok;
begin
 data = {
 mem[addr + 3],
 mem[addr + 2],
 mem[addr + 1],
 mem[addr]
 };
 local_monitor_valid = global_monitor_read(addr);
 exclusive_ok = local_monitor_valid;
end
endtask

// 尝试向内存写入32位数据，并根据全局监视器返回写入成功或失败。始终清除本地监视器标志。

task exclusive_write_32;
 input [31:0] addr;
 input [31:0] data;
 output exclusive_ok;
begin
 if (!local_monitor_valid) begin
 exclusive_ok = 0; // 本地监视器拒绝写入，否则若(global_
 monitor_write(addr)) 成功，则 exclusive_ok = 1;
 // 写入成功
 mem[addr + 3] <= data[31:24];
 mem[addr + 2] <= data[23:16];
 mem[addr + 1] <= data[15: 8];
 mem[addr + 0] <= data[7: 0];
 end else begin
 exclusive_ok = 0; // 全局监视器拒绝写入
 end
 local_monitor_valid = 0; // 始终清除本地监视器标志
end
```

```
endtask
```

函数`global_monitor_read(addr)`和`global_monitor_write(addr)`;上述代码中，这两个函数返回对该地址进行独占读取或写入的全局监视器响应，遵循第2.1.6节中规定的规则。全局监视器确保本hart针对共享相同内存的其他harts执行的读-改-写序列的原子性。

由于Hazard3以硬件顺序的读-改-写重试循环实现了基于AHB5独占的AMO，硬件会在AMO期间将读保留失败提升为存储/AMO故障异常（`mcause = 7`）。该行为避免了访问不支持独占访问的位置时出现无限循环。

以下局部变量适用于所有AMO伪代码：

```
reg done = 0;
reg exclusive_success;
reg [31:0] tmp;
```

#### amoadd.w

以原子方式将寄存器值加至内存，并返回原内存值。

用法：

```
amoadd.w rd, rs2, (rs1)
```

操作：

```
if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = tmp + rs2;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志
```

#### amoand.w

以原子方式将寄存器按位与写入内存。返回原始内存值。

用法：

```
amoand.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = tmp & rs2;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amomax.w**

以原子方式：检查寄存器的有符号值是否大于内存值，若是则写入内存。返回原始内存值。

用法：

```
amomax.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = $signed(tmp) < $signed(rs2) ? rs2 : tmp;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amomaxu.w**

以原子方式：检查寄存器的无符号值是否大于内存值，若是则写入内存。返回原始内存值。

用法：

```
amomaxu.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = tmp < rs2 ? rs2 : tmp;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amomin.w**

原子操作：检查寄存器是否小于内存中的有符号值，若是，则写入内存。返回原内存值。

用法：

```
amomin.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = $signed(tmp) < $signed(rs2) ? tmp : rs2;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amominu.w**

原子操作：检查寄存器是否小于内存中的无符号值，若是，则写入内存。返回原内存值。

用法：

```
amominu.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = tmp < rs2 ? tmp : rs2;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amoor.w**

原子操作：将寄存器的按位或结果写入内存。返回原始内存值。

用法：

```
amoor.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 tmp = tmp | rs2;
 exclusive_write_32(rs1, tmp, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amoswap.w**

原子操作：向内存写入值，并返回写入前内存中的原值。

用法：

```
amoswap.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 exclusive_write_32(rs1, rs2, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**amoxor.w**

原子操作：将寄存器的按位或结果写入内存。返回原始内存值。

用法：

```
amoxor.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因：存储/原子操作地址对齐错误
 done = 1;
end
while (!done) begin
 exclusive_read_32(rs1, tmp, exclusive_success);
 if (!exclusive_success || bus_fault(addr)) begin
 raise_exception(4'h7); // 原因：存储/原子操作访问错误
 done = 1;
 end else begin
 exclusive_write_32(rs1, rs2, done);
 end
end
local_monitor_valid = 0; // 始终清除本地监视器标志

```

**lr.w**

从内存加载值并通过全局监视器建立保留。根据保留是否成功设置本地监视器位。

用法：

```
lr.w rd, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h4); // 原因: 加载地址未对齐
end else if (bus_fault(rs1)) begin
 ise_exception(4'h5); // 原因: 加载故障 end else
begin
 read_exclusive_32(rs1, tmp, local_monitor_valid);
 rd <= tmp;
end

```

**sc.w**

有条件地向内存存储值。仅当本地和全局监视器的保留均有效时，存储操作才算成功。返回 1 表示失败，0 表示成功。

用法：

```
sc.w rd, rs2, (rs1)
```

操作：

```

if (rs1[1:0]) begin
 raise_exception(4'h6); // 原因: 存储/原子操作地址对齐错误
end else if (bus_fault(addr)) begin
 raise_exception(4'h7); // 原因: 存储/AMO故障 end else
 e if (!local_monitor_valid) begin
 rd <= 1; // 被本地监视器拒绝
 end else begin
 write_exclusive_32(rs1, rs2, exclusive_success);
 rd <= !exclusive_success;
 end
 local_monitor_valid = 0; // 始终清除本地监视器标志

```

### 3.8.1.14. C：压缩指令

C 扩展中的所有指令均为其他扩展中 32 位指令的 16 位别名。对于缺少 F 扩展的 Hazard3，这些均为基本 I 指令的别名。它们的行为与对应的 32 位指令完全一致。

C 为 RV32I 中的以下指令添加了压缩别名：

| 字母顺序：先从左到右，后从上到下。 |      |       |      |     |     |
|-------------------|------|-------|------|-----|-----|
| 加                 | addi | and   | andi | beq | bne |
| ebreak            | jal  | jalr  | lui  | lw  | or  |
| slli              | srai | srlti | 减    | sw  | xor |

关于每条指令的压缩限制，请参阅各指令的专门文档。当满足限制条件且为汇编器启用了相关压缩指令扩展时（例如通过在 `-marchISA` 字符串中加入 C），汇编器会自动使用压缩变体。

上述说明同样适用于 Zca 和 Zcb：前者是 C 非浮点子集的别名，后者为 I、M 和 Zbb 扩展中的若干常用指令添加了 16 位别名。每条 Zcmp 指令均扩展为来自 Iextension 的一系列多条指令。

[\(返回索引\)](#)

### 3.8.1.15. Zba：位操作（地址生成）

这些指令用于加速2、4及8字节元素数组的地址生成。若更符合您的需求，也可通过常数3、5及9进行乘法运算。

#### 带1加法

加法操作，第一加数左移1位。

用法：

```
sh1add rd, rs1, rs2
```

操作：

```
rd <= (rs1 << 1) + rs2;
```

#### 带2加法

加法操作，第一加数左移2位。

用法：

```
sh2add rd, rs1, rs2
```

操作：

```
rd <= (rs1 << 2) + rs2;
```

#### 带3加法

加法操作，第一加数左移3位。

用法：

```
sh3add rd, rs1, rs2
```

操作：

```
rd <= (rs1 << 3) + rs2;
```

### 3.8.1.16. Zbb：位操作（基础）

这些指令适用于位域操作及复杂整数算术运算，如软浮点例程中的操作。其中许多指令直接替代常见的RV32I指令对，如 `ext.h → sll, srl`。

**andn**

与第二操作数取反后的按位与操作。

用法：

```
andn rd, rs1, rs2
```

操作：

```
rd <= rs1 & ~rs2;
```

**clz**

计算前导零（从最高有效位开始，向最低有效位方向搜索）。

用法：

```
clz rd, rs1
```

操作：

```
rd <= 32; // 默认值=32, 若无任何置位
reg found = 1'b0; // 局部变量

for (i = 0; i < 32; i = i + 1) begin
 if (rs1[31 - i] && !found) begin
 found = 1'b1;
 rd <= i;
 end
end
```

**cpop**

位计数。

用法：

```
cpop rd, rs1
```

操作：

```
reg [5:0] sum = 6'd0; // 局部变量
for (i = 0; i < 32; i = i + 1)
 sum = sum + rs1[i];
rd <= sum;
```

**ctz**

计算尾随零（从最低有效位开始，向最高有效位方向搜索）。

用法：

```
ctz rd, rs1
```

操作：

```
rd <= 32; // 默认值=32, 若无任何置位
reg found = 1'b0; // 局部变量

for (i = 0; i < 32; i = i + 1) begin
 if (rs1[i] && !found) begin
 found = 1'b1;
 rd <= i;
 end
end
```

**max**

两个数的最大值（有符号）。

用法：

```
max rd, rs1, rs2
```

操作：

```
if ($signed(rs1) < $signed(rs2))
 rd <= rs2;
else
 rd <= rs1;
```

**maxu**

两个数的最大值（无符号）。

用法：

```
maxu rd, rs1, rs2
```

操作：

```
if (rs1 < rs2)
 rd <= rs2;
else
 rd <= rs1;
```

**min**

两个数的最小值（有符号）。

用法：

```
min rd, rs1, rs2
```

操作：

```
if ($signed(rs1) < $signed(rs2))
 rd <= rs1;
else
 rd <= rs2;
```

#### **minu**

两个数的最小值（无符号）。

用法：

```
minu rd, rs1, rs2
```

操作：

```
if (rs1 < rs2)
 rd <= rs1;
else
 rd <= rs2;
```

#### **orc.b**

对每个字节内的位进行按位或组合。生成非零字节掩码。

用法：

```
orc.b rd, rs1
```

操作：

```
rd <= {
 {8{|rs1[31:24]}},
 {8{|rs1[23:16]}},
 {8{|rs1[15:8]}},
 {8{|rs1[7:0]}}
};
```

#### **orn**

与第二操作数取反后的按位或运算。

用法：

```
orn rd, rs1, rs2
```

操作：

```
rd <= rs1 | ~rs2;
```

#### **rev8**

字内字节逆序。

用法：

```
rev8 rd, rs1
```

操作：

```
rd <= {
 rs1[7:0],
 rs1[15:8],
 rs1[23:16],
 rs1[31:24]
};
```

#### **循环左移**

按寄存器值左循环移位，模32。

用法：

```
rol rd, rs1, rs2
```

操作：

```
rd <= ({rs1, rs1} << rs2[4:0]) >> 32;
```

#### **循环右移**

按寄存器右旋，模32。

用法：

```
ror rd, rs1, rs2
```

操作：

```
rd <= {rs1, rs1} >> rs2[4:0];
```

**循环右移带进位**

按立即数右旋。

用法：

```
rori rd, rs1, imm
```

操作：

```
rd <= {rs1, rs1} >> imm;
```

立即数范围： 0至 31。

**符号扩展字节**

从字节符号扩展。

用法：

```
sext.b rd, rs1
```

操作：

```
rd <= {
 {24{rs1[7]}},
 rs1[7:0]
};
```

满足以下条件时可压缩： rd与 rs1匹配，且寄存器位于 x8至 x15之间。

**符号扩展半字**

从半字符号扩展。

用法：

```
sext.h rd, rs1
```

操作：

```
rd <= {
 {16{rs1[15]}},
 rs1[15:0]
};
```

满足以下条件时可压缩： rd与 rs1匹配，且寄存器位于 x8至 x15之间。

**xnor**

与取反操作数的按位异或。等价于按位异或的按位取反。

用法：

```
xnor rd, rs1, rs2
```

操作：

```
rd <= rs1 ^ ~rs2;
```

**zext.b**

从字节零扩展。

用法：

```
zext.b rd, rs1
```

操作：

```
rd <= {
 24'h000000,
 rs1[7:0]
};
```

满足以下条件时可压缩： **rd** 与 **rs1** 匹配，且寄存器位于 **x8** 至 **x15** 之间。

32位操作码 **zext.b** 为andi的伪指令。然而，压缩变体是 **Zcb** 专用指令。它实际上不属于 **Zbb**，但为便于与其他 **sext./ze xt** 指令归类，故此处予以说明。

**zext.h**

从半字节零扩展。

用法：

```
zext.h rd, rs1
```

操作：

```
rd <= {
 16'h0000,
 rs1[15:0]
};
```

满足以下条件时可压缩： **rd** 与 **rs1** 匹配，且寄存器位于 **x8** 至 **x15** 之间。

### 3.8.1.17. Zbs: 位操作（单个位）

这些指令用于反转、设置、清除及提取寄存器中的单个位。

#### bclr

清除单个位。

用法：

```
bclr rd, rs1, rs2
```

操作：

```
rd <= rs1 & ~(32'h1 << rs2[4:0]);
```

#### bclri

清除单个位（立即数）。

用法：

```
bclri rd, rs1, imm
```

操作：

```
rd <= rs1 & ~(32'h1 << imm);
```

立即数范围： 0至 31。

#### bext

提取单个位。

用法：

```
bext rd, rs1, rs2
```

操作：

```
rd <= (rs1 >> rs2[4:0]) & 32'h1;
```

#### bexti

提取单个位（立即数）。

用法：

```
bexti rd, rs1, imm
```

操作:

```
rd <= (rs1 >> imm) & 32'h1;
```

立即数范围: 0至 31。

**binv**

反转单个位。

用法:

```
binv rd, rs1, rs2
```

操作:

```
rd <= rs1 ^ (32'h1 << rs2[4:0]);
```

**binvi**

反转单个位 (立即数)。

用法:

```
binvi rd, rs1, imm
```

操作:

```
rd <= rs1 ^ (32'h1 << imm);
```

立即数范围: 0至 31。

**bset**

设置单个位。

用法:

```
bset rd, rs1, rs2
```

操作:

```
rd <= rs1 | (32'h1 << rs2[4:0])
```

**bseti**

设置单个位 (立即数)。

用法:

```
bseti rd, rs1, imm
```

操作：

```
rd <= rs1 | (32'h1 << imm);
```

立即数范围： 0 至 31。

### 3.8.1.18. Zbkb：用于密码学的基本位操作

Zbkb 与 Zbb（基本位操作）有较大重叠。本节涵盖 Zbkb 中的指令，但不包括 Zbb 中的指令。

**brev8**

在每个字节内进行位反转操作。

用法：

```
brev8 rd, rs1
```

操作：

```
for (i = 0; i < 32; i = i + 8) begin
 for (j = 0; j < 8; j = j + 1) begin
 rd[i + j] <= rs1[i + (7 - j)];
 end
end
```

**pack**

将两个半字合并为一个字。

用法：

```
pack rd, rs1, rs2
```

操作：

```
rd <= {
 rs2[15:0],
 rs1[15:0]
};
```

**打包高**

将两个字节合并为一个半字。

用法：

```
packh rd, rs1, rs2
```

操作：

```
rd <= {
 16'h0000,
 rs2[7:0],
 rs1[7:0]
};
```

### 解压

将寄存器中的奇偶位解交织至结果的上下半部分。

用法：

```
unzip rd, rs1
```

操作：

```
for (i = 0; i < 32; i = i + 2) begin
 rd[i / 2] <= rs1[i];
 rd[i / 2 + 16] <= rs1[i + 1];
end
```

### zip

将寄存器的上下半部分交织为结果的奇偶位。

用法：

```
zip rd, rs1
```

操作：

```
for (i = 0; i < 32; i = i + 2) begin
 rd[i] <= rs1[i / 2];
 rd[i + 1] <= rs1[i / 2 + 16];
end
```

### 3.8.1.19. Zcb：新增基础压缩指令

Zcb为来自 I、M 和 Zbb 扩展的以下指令新增了16位压缩别名：

| 字母顺序：先从左到右，后从上到下。 |    |     |     |     |     |
|-------------------|----|-----|-----|-----|-----|
| lbu               | lh | lhu | mul | not | 设置位 |

|                   |        |       |        |        |  |
|-------------------|--------|-------|--------|--------|--|
| 字母顺序：先从左到右，后从上到下。 |        |       |        |        |  |
| 符号扩展字节            | 符号扩展半字 | 带符号右移 | zext.b | zext.h |  |

请参阅各指令文档，了解每条指令的可压缩性限制。

[\(返回索引\)](#)

### 3.8.1.20. Zcmp：压缩的推送、弹出和双重移动

Zcmp增加了16位指令，这些指令可扩展为用于函数序言和尾声的常见32位RV32I指令序列。以下是可用指令的简要描述：

- **cm.push**：分配栈帧并保存寄存器。
  - 将 **ra** 推入栈中。
  - 可选择依次从 **s0** 到 **s11** 推入多个保存寄存器，顺序从 **s0** 开始。
  - 将栈指针的总减少量四舍五入至16字节的整数倍，以在已对齐时保持栈对齐。
  - 将栈指针额外递减最多48字节，按16字节倍数递减，以分配附加的栈帧空间。
  - 共有十二个 **s\*** 寄存器，您可以推入任意数量，唯独不能推入寄存器11。若需推入超过十个 **s\*** 寄存器，则必须推入十二个。
- **cm.pop**：**cm.push**的逆操作。释放栈帧并恢复 **ra**，可选地恢复 **s0** 到 **s11** 寄存器。
- **cm.popret**：等同于先执行 **cm.pop**，然后执行 **ret**。释放栈帧，恢复被保存的寄存器，并返回。
- **cm.popretz**：等同于执行 **cm.pop; li a0, 0; ret**。函数通常返回常量 **0**。
- **cm.mvsa01**：将 **a0** 和 **a1** 移动到 **s0** 到 **s7** 范围内的任意两个寄存器。用于在嵌入式调用过程中保存参数。
- **cm.mva01s**：将 **s0** 到 **s7** 范围内任意两个寄存器的值移动到 **a0** 和 **a1**。用于恢复已保存的参数。

有关堆栈布局以及中断相关的原子性等关键细节，请参见第3.8.1.1节中有关 Zcmp 规范的链接。有关 Hazard3 处理器中这些指令的周期计数，请参见第3.8.7节。

[\(返回索引\)](#)

### 3.8.1.21. RV32I 与 Zifencei：内存排序指令

这些指令控制多核系统中负载与存储操作的可观察内存顺序。它们还强制保证某核的指令取指能够观察到其自身的存储。

#### fence

根据该核的程序顺序，限制该核访问在整个内存顺序中的位置。

用法：

```
// <set> 为匹配正则表达式 i?o?r?w? 的非空字符串
fence <set>, <set> // 前驱, 后继
fence // 伪代码: fence iorw, iorw
fence.tso // fence rw, rw 的变体; 详见下文
```

操作：Hazard3 无存储缓冲区，并假定内存子系统为顺序一致。因此，无需额外记录管理以保证共享内存的顺序，该指令作为无操作执行。（SDK 仍使用 **fence** 指令及其有序变体 **amo\*.w**，以保证跨平台的兼容性）

利用放宽的内存排序。

名义上，`fence`屏障强制要求前驱集合在整体内存顺序中出现在后继集合之前。这些集合分别包含该hart在程序顺序中`fence`指令之前和之后的内存访问，并各自通过4位掩码进一步过滤：

- 设备输入 (I)
- 设备输出 (O)
- 读取 (R)
- 写入 (W)

`fence.tso`（总存储顺序）变体等同于`fence rw, rw`，只是不强制写前于读的顺序。

#### `fence.i`

指令屏障。确保该hart上随后的指令取指能观察到该hart先前的存储操作。

用法：

```
fence.i
```

操作：

1. 清除分支目标缓冲区（第3.8.7.10节）
2. 跳转至顺序下一条指令地址（`pc + 4`），以清除预取缓冲区。

预取缓冲区可能会使指令取指操作与程序顺序更早的存储操作重排序。例如：

```
la a0, label // 获取存储指令的地址
li a1, 0x9002 // 获取 c.ebreak 的立即数值
div t1, t1, t1 // 长时间执行的指令，填充预取缓冲区
sh a1, (a0) // 写入下一个地址。（16位操作码）
label:
nop // （16位操作码）
```

如果您在 Hazard3 上执行上述代码，可能会在 `label` 处产生断点异常，也可能不会。结果取决于总线访问所需的周期数。此行为符合 RISC-V 内存模型的规定。

此情况通常仅在顺序执行时才会发生，因为 Hazard3 不会通过控制流指令进行预取，除非是分支目标缓冲区中当前分配的向后条件分支。特别是，它不会通过类似 `ret` 的间接分支进行预取。您在实际中不大可能遇到此问题；

但请注意，`fence.i` 是解决此类问题的标准方法。

如果您写入当前在分支目标缓冲区标记的条件分支指令地址，并且未先执行`fence.i`便执行该条件分支指令，Hazard3 的行为将无法预测。请通过始终在写入内存与执行该内存之间执行`fence.i`指令以避免此类问题。

### 3.8.1.22. Zicsr：控制及状态寄存器访问

这些指令访问第3.8.9节所列控制与状态寄存器（CSR）。CSR指令可用于读取CSR、修改CSR，或同时读取并修改同一CSR。修改包括普通写入、原子位清除或原子位设置。

CSR地址范围为 `0x000` 至 `0xffff`（12位，共4096个可能的CSR）。CSR地址为指令中的立即数，故无法使用运行时值进行索引。汇编器接受数字常量或

诸如 `mstatus` 的CSR名称作为CSR地址。

#### `csrrc`

同时读取并清除CSR中的位。

用法：

```
csrrc rd, <addr>, rs1
csrc <addr>, rs1 // 伪指令：rd 为零
```

操作：

```
rd <= csr[addr];
if (regnum_rs1 != 5'h00)
 csr[addr] <= csr[addr] & ~rs1;
```

#### `csrrci`

同时读取并清除CSR中的位，清除操作使用立即数。

用法：

```
csrrci rd, <addr>, imm
csrci <addr>, imm // 伪指令：rd 为零
```

操作：

```
rd <= csr[addr];
if (imm != 32'h0)
 csr[addr] <= csr[addr] & ~imm;
```

立即数范围： 0 至 31。

#### `csrrs`

同时读取并设置CSR中的位。

用法：

```
csrrs rd, <addr>, rs1
csrs <addr>, rs1 // 伪指令：rd 为零
csrr rd, <addr> // 伪代码：rs1为零
```

操作：

```
rd <= csr[addr];
if (regnum_rs1 != 5'h00)
 csr[addr] <= csr[addr] | rs1;
```

**csrrsi**

同时读取并设置CSR中的位，设置操作使用立即数。

用法：

```
csrrsi rd, <addr>, imm
csrsi <addr>, imm // 伪指令：rd 为零
```

操作：

```
rd <= csr[addr];
if (imm != 32'h0)
 csr[addr] <= csr[addr] | imm;
```

立即数范围： 0 至 31。

**csrrw**

同时读取并写入CSR。

用法：

```
csrrw rd, <addr>, rs1
csrw <addr>, rs1 // 伪指令：rd 为零
```

操作：

```
if (regnum_rd != 5'h00)
 rd <= csr[addr];
csr[addr] <= rs1;
```

**csrrwi**

同时读取并写入CSR，写入时使用立即数。

用法：

```
csrrwi rd, <addr>, imm
csrwi <addr>, imm // 伪指令：rd 为零
```

操作：

```
if (regnum_rd != 5'h00)
 rd <= csr[addr];
csr[addr] <= imm;
```

立即数范围： 0 至 31。

### 3.8.1.23. 特权指令

这些指令属于特权ISA手册中定义的陷阱与中断控制支持。该支持的另一部分为CSR（参见第3.8.9节）。

#### **ebreak**

引发断点异常。

用法：

```
ebreak
```

操作：

```
raise_exception(4'h3); // Cause = ebreak
```

可压缩条件：始终可压缩。

特权级别要求：任意等级。

有关RISC-V陷阱入口序列的详细说明，参见第3.8.4节。所有异常均在Hazard3的M态下陷阱处理。异常程序计数器 **me\_pc** 指向 **ebreak** 指令的起始地址。

外部调试主机可以捕获断点指令的执行。如果内核处于M模式且DCSR.EBREAKM已设置，内核将进入调试模式，而非触发异常。在U模式下，DCSR.EBREAKU启用相同的行为。

#### **ecall**

环境调用。引发异常以访问更高特权级的处理程序。

用法：

```
ecall
```

操作：

```
if (priv == 2'h3)
 raise_exception(4'hb); // 原因：来自M模式的环境调用，否则

 raise_exception(4'h8); // 原因：来自U模式的环境调用
```

特权级别要求：任意等级。

有关RISC-V陷阱入口序列的详细说明，参见第3.8.4节。所有异常均在Hazard3中陷入M模式。异常程序计数器 **me\_pc** 指向 **ecall** 指令的起始位置。

#### **mret**

从M模式陷阱返回。

用法：

```
mret
```

操作：执行第3.8.4节中描述的陷阱返回序列。

特权要求：仅限M模式。

wfi

等待中断。

用法：

```
wfi
```

操作：暂停执行，直到处理器被中断或进入调试模式。

特权要求：始终允许M模式。当 MSTATUS.TW 清零时，允许进入用户模式。

wfi 指令会忽略全局中断使能标志 MSTATUS.MIE，但会遵循所有其他中断控制。例如：

- 若 MIP.MEIP 为 1，MIE.MEIE 为 1，且 MSTATUS.MIE 为 0，则 wfi 指令会立即执行下去，无需暂停。
- 在此示例中，将 MSTATUS.MIE 设为 1 将使核心立即响应中断。
- 当 MIP 和 MIE 中均无置位标志时，wfi 指令会等待，直到至少有一个置位标志出现。

当 wfi 指令被中断时，异常返回地址 MEPC 指向紧跟该 wfi 指令之后的下一条指令。

当调试器在 wfi 指令执行期间暂停核心时，DPC 指向紧跟该 wfi 指令之后的下一条指令。wfi 在单步指令执行模式下作为空操作执行（不会引起暂停），且在程序缓冲区的调试模式下亦是如此。

Hazard3 的 MSLEEP CSR 控制核心在 wfi 睡眠状态下可执行的额外省电措施。

### 3.8.2. 内存访问

Hazard3 访问位于 4 GB (2<sup>32</sup>字节) 物理地址空间内的内存。不存在地址转换。整型寄存器的每个可能值唯一标识物理地址空间中的一个字节。多字节数据占据连续的字节地址。

#### 3.8.2.1. 字节序

Hazard3 对所有加载与存储访问始终采用小端字节序。RISC-V 指令获取始终采用小端字节序。

这意味着在如 sw 指令（传输四个字节）等多字节访问中，存储于较高字节地址的数据具有更大数值权重。例如：

```
li a0, 0x0d0c0b0a // 在寄存器中生成常数
la a4, some_global_variable // 生成地址（假设地址 % 4 == 0）
sw a0, (a4) // 对内存进行4字节写入
lbu a0, 0(a4) // 从地址偏移0加载字节: 0x0a
lbu a1, 1(a4) // 从地址偏移1加载字节: 0x0b
lbu a2, 2(a4) // 从地址偏移2加载字节: 0x0c
lbu a3, 3(a4) // 从地址偏移3加载字节: 0x0d
```

### 3.8.2.2 物理存储器属性

RP2350地址空间具有以下物理存储器属性：

表366 RP2350地址空间物理存储器属性列表  
主SRAM支持所有原子操作，其他地址不支持。

外设为非幂等，其他所有地址均为幂等。

| 起始         | 结束         | 描述           | 访问权限        | 原子性                            | 幂等性 |
|------------|------------|--------------|-------------|--------------------------------|-----|
| 0x00000000 | 0x00007fff | 启动只读存储器      | 无AMO        | RsrvNone, AMONone              | 幂等  |
| 0x10000000 | 0x13fffff  | XIP, 缓存      | 无AMO        | RsrvNone, AMONone              | 幂等  |
| 0x14000000 | 0x17fffff  | XIP, 非缓存     | 无AMO        | RsrvNone, AMONone              | 幂等  |
| 0x18000000 | 0x1bfffff  | XIP, 缓存维护    | 仅写          | RsrvNone, AMONone              | 幂等  |
| 0x1c000000 | 0x1fffffff | XIP, 非缓存+未翻译 | 无AMO        | RsrvNone, AMONone              | 幂等  |
| 0x20000000 | 0x20081fff | 主SRAM        | 任意          | RsrvNonEventual, AMOArithmetic | 幂等  |
| 0x40000000 | 0x4fffffff | APB 外设       | 无AMO, 无指令获取 | RsrvNone, AMONone              | 非幂等 |
| 0x50000000 | 0x5fffffff | AHB 外设       | 无AMO, 无指令获取 | RsrvNone, AMONone              | 非幂等 |
| 0xd0000000 | 0xdfffffff | SIO 外设       | 无AMO, 无指令获取 | RsrvNone, AMONone              | 非幂等 |

所有地址均具有强序性。表 366 中未列出的任何地址均为空闲地址。访问这些地址除返回总线错误外，不产生任何其他效果。

Hazard3 的 PMP 实现要求非只读幂等的 PMA 也必须为不可执行，因为其执行权限在指令执行点而非指令获取点强制执行。因此，表 366 中所有非幂等的位置均为不可执行。此要求在低于 PMP 级别实现，任何特权级别执行这些地址均必然产生异常。

从 PMA 角度看，缓存的 XIP 区域不可缓存，因为该缓存为内存控制器私有。

每个系统地址由单一缓存控制器服务或无服务，因此各核间一致性无关紧要。在执行诸如闪存编程之类的操作后，您可能需要进行手动缓存维护，但这是 XIP 子系统的细节，而非系统级内存模型的内容。

有关这些属性的定义，请参见第3.8.1.1节所链接的RISC-V特权ISA手册第3.6节。

### 3.8.3. 内存保护

Hazard3实现了物理内存保护（PMP）。它未实现Sv32虚拟内存扩展及其相关保护机制。

PMP定义了物理地址的权限。它主要保护M模式内存免受S模式和U模式的访问。Hazard3仅实现了M模式和U模式。

PMP区域适用于一段字节地址范围的读、写和执行权限。每个区域设有一个地址寄存器，分别为PMPADDR0至PMPADD R15，以及一个封装于PMPCFG0至PMPCFG3中的8位配置字段。对U模式的读、写和执行权限始终被强制执行。根据各区域的PMPCFG L位及PMPCFGM0寄存器的设置，该权限也可能被强制应用于M模式。

RP2350 配置 Hazard3 的 PMP 硬件，具备以下特点：

- 8×动态可配置区域，**0**至**7**
- 3×静态配置（硬连线）区域，**8**至**10**
- （剩余区域**11**至**15**被硬连线设置为关闭）
- 粒度为32字节
- 仅支持自然对齐的2的幂次方区域形状（**NAPOT**）
- 自定义PMPCFGM0 CSR可对单个区域施加M模式权限，且不会锁定

第3.8.8.1节定义了硬连线区域**8**至**10**的配置。这些区域对RP2350只读存储器和外设应用默认的U模式权限，以避免使用动态区域覆盖这些地址。系统级ACCESSCTRL寄存器（第10.6节）可分别将每个外设分配为M模式或U模式。

当多个PMP区域匹配同一字节地址时，以编号最低的区域优先生效。其他区域将被忽略。

### 3.8.3.1. PMP地址寄存器

PMP地址寄存器PMPADDR0至PMPADDR15中的地址经过右移两位存储，因此在启用Sv32地址转换时，可覆盖16GB的物理地址空间。Hazard3不实现地址转换，故物理地址空间为4GB（32位字节寻址），且每个地址寄存器的两个最高有效位硬连线为零。

Hazard3的RP2350配置仅支持**OFF**和**NAPOT**两种PMPCFGA字段值（例如PMPCFG0.R0\_A）。将**A**设置为**OFF**表示该区域不匹配任何字节，实质被禁用。将**A**设置为**NAPOT**表示该区域匹配自然对齐的一段字节（基址模该大小为零），且大小为二的幂。

PMP地址值中尾随的1的数量编码了一个NAPOT区域的大小。这是从最低有效位开始连续计数的1的数量，直到遇到0为止。没有尾随1（以0结尾）的PMP地址值对应一个大小为八字节的区域，且每增加一个1位时区域大小翻倍。

PMP区域匹配最低有效0位左侧的地址位。由于PMP地址寄存器右移两位，比较时地址也必须进行相同的右移。以下示例演示如何基于PMPADDRx值匹配地址：

- 30位全为一的位模式**0xffffffff**具有最大可能的大小，并能匹配所有地址。
- 全为零的位模式**0x00000000**具有最小可能的大小。
  - 由于没有尾随的1，此匹配从PMP地址寄存器的第1位开始。
  - 由于地址右移两位，此地址范围为从地址**0x0**开始的八字节区域。
- 位模式**0x????????**（其中？为任意数字）匹配任意64字节区域。
  - 将此64字节区域的基址右移两位，即可获得PMPADDRx值的29:4位。
- 位模式**0x0800000f**匹配字节地址范围在**0x20000000**至**0x2000007f**之间，即SRAM的前128字节。
  - 将基地址（**0x20000000**）右移两位，得到**0x08000000**。
  - 通过添加尾随的1位以增大区域大小，最终得到值**0x0800000f**。
  - 该区域大小等于八字节乘以尾随1位数的2的幂，此处四个尾随1位，计算结果为 $8 \times 2^4 = 128$ 字节。

有关更多PMP地址匹配模式的示例，请参见第3.8.8.1节中硬连线的PMP区域值。

RP2350将Hazard3配置为32字节的粒度。这表示当该区域启用时，每个PMP地址寄存器的最低两位被硬连线为全1。硬件不会解码小于32字节的地址区域。

### 3.8.3.2. PMP权限

每个8位PMP配置字段包含三个权限标志：

- **R** 允许非指令取指的读取操作：
  - 加载指令
  - AMO的读取阶段
- **W** 允许写入操作：
  - 存储指令
  - AMO的写入阶段
- **X** 允许指令执行

每个权限值为**1**表示授予，值为**0**表示撤销。这些权限适用于U模式对该区域的访问。当满足下列任一条件时，亦适用于M模式访问：

- **L**（锁定）配置位为**1**
- 该区域的Hazard3自定义PMPCFGM0寄存器位为**1**

**L**（锁定）位还会锁定关联的PMP地址寄存器和8位PMP配置字段，从而忽略后续的写入操作。您应始终从区域0开始连续锁定PMP区域，以防止锁定区域被未锁定区域绕过。

不匹配任何PMP区域的U态访问无权限：所有内存访问均将失败。不匹配任何PMP区域的M态访问具有全部权限。第3.8.8.1节中硬连线的PMP区域定义了只读存储器和外设地址范围的额外U态权限：启用任一动态配置区域后，该权限可被覆盖。

#### ① 注意

由于RP2350-E6，PMP配置字段中的字段顺序为 **R、W、X**（最高有效位优先），而非标准的 **X、W、R**。SDK寄存器头文件与实际实现顺序一致。

### 3.8.3.3 跨越多个PMP区域的访问

Hazard3不支持非自然对齐的加载或存储，尝试时仅产生标准异常。由于 **NAPOT**PMP区域始终自然对齐，加载或存储操作不可能跨越两个PMP区域。因此，一条加载或存储指令所覆盖的所有字节，最多由一个与该指令访问的最低字节地址匹配的活动PMP区域确定。

指令最大长度为32位，对齐要求最低16位。因此，指令有可能匹配多个PMP区域。发生这种情况时，指令会产生指令异常（**mcause = 0x1**），除非存在编号更低且完全覆盖该指令的PMP区域。编号较低的PMP区域优先。

特权ISA规范中相关表述为：“匹配访问任一字节的编号最低的PMP条目，决定该访问是否成功。” 匹配的PMP条目必须匹配访问的所有字节，否则访问将失败，无论L、R、W和X位的状态如何。”（摘自RISC-V特权ISA手册，版本20211203，第60页）

RISC-V规范对内存保护检查中认定的单次访问范围具有一定灵活性。Hazard3视单条指令的取指为一次完整访问。因此，即便两个PMP区域均授予执行权限，也禁止跨越两个PMP区域的指令取指。基于该架构规则，便携式RISC-V软件不得假定可执行跨越多个PMP区域的指令。

建议通过采用*hole-punching*区域配置，而非*glueing*配置，以避免此类问题。假设您要覆盖SRAM的前12 kB（**0x20000000 → 0x20002fff**），可通过以下两种方式实现：

- 一个区域向 **0x2**添加权限 **0000000 → 0x20001fff**，另一个区域向 **0x20002000 → 0x20002fff**

- 一个区域向 `0x2`添加权限 `0000000 → 0x20003fff`, 且编号较低的区域从 `0x20003000 → 0x20003ffff` 移除权限

前者在两个区域之间存在缝隙, 可能对某些平台产生不良影响。后者则完全避免了该问题。

### 3.8.4. 中断与异常

在RISC-V特权ISA手册中, `trap`指中断或异常:

#### 中断

处理器外部发送的信号, 要求处理器暂时中止当前任务以处理某些系统级事件。处理器通过转移控制至中断处理函数来响应该信号。

#### 异常

指令遇到某种条件, 导致该指令无法正常完成。处理器将控制权转移至异常处理函数, 以处理异常情况, 之后方可恢复执行。

这两者密切相关, 统称为陷阱, 以避免重复陈述所有内容。

硬件在进入陷阱时自动且以原子方式执行以下步骤:

1. 将被中断或发生异常的指令地址保存至 MEPC
2. 设置 MCAUSE 的最高有效位以指示原因为中断, 或清除该位以表示异常
3. 将详细的陷阱原因写入 MCAUSE 寄存器的最低有效位
4. 将当前特权级保存到 MSTATUS.MPP
5. 将特权级设置为 M 模式 (注意 Hazard3 不支持 S 模式)
6. 将 MSTATUS.MIE 的当前值保存至 MSTATUS.MPIE
7. 通过清除 MSTATUS.MIE 禁用中断
8. 根据陷阱原因跳转至 MTVEC 的正确偏移地址

#### 注意

上述事件序列为标准流程, 亦记载于 RISC-V 特权 ISA 手册。RISC-V 规范的链接列表详见第 3.8.1.1 节。

MEPC 指向的指令之前的所有指令均正常执行, 其效果对陷阱处理程序可见。早期指令不受异常或中断的影响。另一方面, 由MEPC指向的指令及其之后的所有指令, 在进入陷阱处理程序之前不会执行。这些指令没有可见的副作用, 但在加载/存储故障异常情况下, 总线故障本身可能对总线或外围设备产生可感知的影响。

从体系结构角度进一步说明MEPC的行为, 所有陷阱均为精确陷阱, 即存在程序顺序中的节点, 陷阱处理程序观察到所有之前指令均已完成退役, 而所有之后指令尚未执行。MEPC寄存器指示该节点。所有异常亦为同步异常, 意味着存在特定指令引发陷阱, 陷阱在该指令与其前序指令之间的程序顺序处发生。

M态软件通过执行 `mret` 指令, 在处理程序结束时返回到被中断或引发异常的指令。此过程在很大程度上逆转了进入陷阱的步骤:

1. 将核心特权级恢复到 MSTATUS.MPP 的值
2. 向 MSTATUS.MPP 写入 `0` (用户模式)

### 3. 从 MSTATUS.MPIE 恢复 MSTATUS.MIE

4. 向 MSTATUS.MPIE 写入 1

5. 跳转至 MEPC 中的地址。

通常，退出时恢复的值与进入时保存的值完全相同。但情况不一定如此，因为上述所有 CSR 均可由机器模式软件随时读写。手动操作陷阱处理 CSR 对于上下文切换等低级操作系统功能非常有用，或者通过在返回前递增 MEPC，使异常处理程序返回至陷阱点之后的指令。例如，当首次从机器模式进入用户模式代码时，可执行 mret 而无需事先发生陷阱。

硬件不会保存或恢复任何其他寄存器。特别是硬件不会保存核心通用寄存器，软件必须确保处理程序的执行不会干扰前台上下文。对于中断，这可能意味着在被中断者的堆栈上保存核心寄存器，或使用 MSCRATCH CSR 交换堆栈指针，然后在专用的中断堆栈上保存寄存器。对于致命异常来说，这可能无关紧要，因为处理程序不要求返回。

#### 3.8.4.1. 异常

异常发生的原因多种多样。MCAUSE 指示最近异常的具体原因：

| 原因  | 含义                                                                                               |
|-----|--------------------------------------------------------------------------------------------------|
| 0x0 | 指令对齐错误：RP2350 不会发生此异常，因为实现了 16 位压缩指令，且不可跳转到非字节对齐地址。                                              |
| 0x1 | 指令取指错误：尝试从不支持指令取指的地址（如 RP2350 上的 APB/AHB 外设）、缺乏 PMP 执行权限、被 ACCESSCTRL 禁止，或内存设备本身返回错误。            |
| 0x2 | 非法指令：遇到未被该处理器实现的有效 RISC-V 操作码指令，或尝试访问不存在的 CSR，或尝试执行特权指令或访问特权 CSR 时权限不足。                          |
| 0x3 | 断点：执行了 ebreak 或 c.ebreak 指令，但未被任何外部调试主机捕获（DCSR.EBREAKM 或 DCSR.EBREAKU 未设置）。                      |
| 0x4 | 加载对齐错误：尝试从非访问大小倍数的地址进行加载。                                                                        |
| 0x5 | 加载错误：尝试从不存在的地址加载数据，或缺乏 PMP 读取权限，或被 ACCESSCTRL 禁止，或外设返回错误。                                        |
| 0x6 | 存储/AMO 对齐错误：尝试写入非访问大小倍数的地址。                                                                      |
| 0x7 | 存储/AMO 错误：尝试写入不存在的地址，或缺乏 PMP 写入权限，或被 ACCESSCTRL 禁止，或外设返回错误。尝试对不支持 AHB5 排他访问的地址执行 AMO 操作时亦会触发此错误。 |
| 0x8 | 执行了 U 模式下的 ecall 指令。                                                                             |
| 0xb | 执行了 M 模式下的 ecall 指令。                                                                             |

异常无论原因如何，无论是否启用了中断向量，都精确跳转至 MTVEC 的地址。

MSTATUS.MIE 全局中断使能不影响异常的进入。即使异常被禁用，仍然可以捕获异常并陷入异常处理程序。

从异常返回将跳转至 MEPC，硬件在进入异常处理程序前会将其设置为引发异常的指令地址。这意味着默认情况下，你将返回到导致该异常的完全相同指令处。仿真非法指令时，应在返回前递增 mepc，以使执行从出错指令之后继续。

Hazard3 将 mtval 固定硬接为零。要仿真未对齐的加载/存储指令，必须解码该指令并

读取溢出的寄存器状态以计算地址；要仿真非法指令，必须通过解引用 `mepc` 自行从内存读取指令位。

### 3.8.4.2. 中断

Hazard3实现了标准RISC-V中断方案，包含一个外部中断路由至MIP.MEIP，标准计时器和软件中断分别路由至MTIP和MSIP。可在外部集成标准RISC-V PLIC等中断控制器，通过单个外部中断线路由多个中断。或者，Hazard3中断控制器（参见Xh3irq扩展，3.8.6.1节）将多个外部中断复用至MIP.MEIP，使中断能够高效相互抢占，并可针对各中断配置动态优先级。

RP2350配置了配备Xh3irq中断控制器的Hazard3，具备52条外部中断线及16级抢占优先级。系统级中断的IRQ编号（详见3.2节）在Arm与RISC-V架构中保持一致。

仅当以下所有条件均成立时，核心才进入中断状态：

- MSTATUS.MIE位被置位
- 标准MIP CSR中的某个待处理中断位被置位
- 标准MIE CSR中对应的中断使能位亦被置位

向量化禁用时（MTVEC最低有效位清零），中断控制直接转移至 `mtvec` 指定的地址。

将最低有效位设为1则启用向量化：中断控制转移至地址`mtvec + 4 * cause`，其中中断原因包括：

- `meip`: 原因 = 11
- `mtip`: 原因 = 7
- `msip`: 原因 = 3

写入 `mtvec` 的指针必须是字对齐（4字节）。此外，启用矢量模式时，指针必须对齐到表大小，并向上取整为2的幂次方，即64字节对齐。在RP2350上，`mtvec`除第1位外完全可写，第1位为硬连线0，仅用于Hazard3不支持的额外矢量模式。

当多个中断同时激活时，硬件将按顺序 `meip` > `msip` > `mtip` 选择一个进入。（该顺序与cause值的顺序不完全相同。）

#### 3.8.4.2.1. RISC-V中断信号

标准定时器中断MIP.MTIP连接至SIO子系统中的RISC-V平台定时器（第3.1.8节）。该定时器为64位，且每核带有64位比较值。当定时器值大于或等于比较值时，中断被触发；当值小于比较值时，中断自动取消。同一中断信号同时出现在系统级IRQ中，标识为 `SIO_IRQ_MTIMECMP`(IRQ 29)。该定时器为标准RISC-V外设，操作系统常用其生成上下文切换中断。

标准软件中断MIP.MSIP连接至SIO子系统中的RISCV\_SOFTIRQ寄存器。该寄存器每个hart对应一个位，用于向该hart触发软IRQ中断。此功能可用于中断其他hart，或模拟其他hart中断自身，从而使中断处理代码更加对称。在RP2350中，hart与核心一一对应，因此等同于每个核心拥有一个软IRQ。

Hazard3的内部中断控制器依据其内部状态和系统级中断信号驱动MIP.MEIP外部中断待处理位，在安全且必要时切换到中断向量。第3.8.6.1节详细描述了Xh3irq中断控制器。

### 3.8.4.2.2. 中断调用约定

默认的SDK `hardware_irq` 库要求系统级IRQ注册的函数指针必须为普通C函数。

传入诸如 `set_exclusive_irq_handler()` 等函数的中断处理程序不得带有 `__attribute__((interrupt))` 属性。这一API细节在SDK支持的所有架构中保持一致。使用常规C调用约定在高中断负载下同样高效，因为保存和恢复所有调用者保存寄存器及临时寄存器的开销可通过多个中断处理程序间的尾部共享加以摊销，且由低优先级IRQ触发的保存操作可被在保存过程中断的高优先级IRQ接管。

相反，通过SDK的 `irq_set_riscv_vector_handler()` 函数为标准RISC-V `mtip` 和 `msip` 中断注册的处理程序必须标注 `__attribute__((interrupt))` 属性。就生成的代码而言，应采用边调试边保存（`save-as-you-go`）调用约定，并以 `mret` 指令结束。这些中断由硬件直接触发，无需任何中间调度代码。

由于软件负责从 `meip` 向量调度至各系统中断处理程序，因此可以通过提供不同的调度实现，支持其他中断调用约定。

### 3.8.5. 调试

#### RISC-V 调试规范

Hazard3 实现了 RISC-V 外部调试支持规范 0.13.2 版本，详见：

[riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf](https://riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf)

RP2350 实现了单一 RISC-V 调试模块，支持对两个 Hazard3 处理器实例进行调试访问。任何实现了 RISC-V 外部调试支持规范 0.13.2 版本的调试转换器均应支持 Hazard3，本文为完整性目的描述了其中一些实现定义的行为细节。调试模块源代码可在 Hazard3 仓库中查阅，以便解答有关调试实现的更详细问题。

如在 RP2350 上配置，Hazard3 支持以下标准 RISC-V 调试功能：

- 每个处理器的运行／停止／复位控制
- 所有处理器的复位停止支持
- Hart 阵列掩码寄存器，用于同时停止／恢复多个处理器
- 对通用寄存器（GPR）的抽象访问
- 程序缓冲区：2字，带有隐式的 `ebreak`（`impebreak`）
- 抽象命令的自动触发（`abstractauto`）
- 系统总线访问，与核心1的加载／存储端口仲裁
- 具有4个硬件断点的指令地址触发单元

#### 3.8.5.1. 访问调试模块

调试模块通过 CoreSight APB-AP 访问，访问方式有两种：

- 外部访问，通过系统的 SW-DP（见第 3.5 节）
- 内部访问，通过自托管调试（见第 3.5.6 节）

调试模块的 APB-AP 位于调试地址空间偏移 `0xa000` 处。调试模块起始于 APB-AP 下游地址空间的地址 0。调试模块以四字节为单位访问寄存器，因为 APB 是按字节寻址而非按字寻址。这意味着 RISC-V 调试规范中列出的调试模块寄存器地址必须乘以 4。

### 3.8.5.2 硬线程

每个Hazard3核心恰好拥有一个硬件线程，亦称为 **hart**。这表示每个处理器一次仅执行单个指令流。RP2350上的两个Hazard3处理器核心，编号为0和1的核心，分别对应hart ID 0和1。这些值可从每个处理器的MHARTID寄存器读取，且与SI0中CPUID寄存器的读取值一致。

RP2350调试模块中的`dmcontrol.hartsel`字段仅支持写入0和1（该字段仅实现一个可写位），对应于hart ID 0和1，分别在核心0和核心1上执行。

### 3.8.5.3. 复位

`dmcontrol.hartreset`字段仅复位所选核心。这可以是由`dmcontrol.hartsel`选择的单个核心，也可以是由`hart`数组掩码选择的多个核心。它不会复位未选择的核心，也不会复位其他任何系统硬件。该系统中，`hart`与核心之间存在一一对应关系。

`dmcontrol.ndmreset`字段复位两个核心。它不会复位任何其他硬件。根据规范：“该复位具体影响的内容由实现决定，只要能够从执行的第一条指令开始调试程序即可。”

### 3.8.5.4. 实现定义的行为

以下内容未被实现：

- 抽象访问内存
- 抽象访问控制状态寄存器（CSR）
- 递增型抽象访问通用寄存器（GPR）

核心行为如下：

- 在调试模式下，`branch`、`jal`、`jalr`和`auipc`均为非法指令，因为它们需观察程序计数器（PC）；执行时将导致暂停。程序缓冲区执行并在`abstractes.cmderr`报告异常
- `dret`指令未实现（使用专用的DM到核心信号来指示恢复）
- `dscratchCSR`未实现
- 调试模块的`data0`寄存器映射为核心中的CSR，称为DMDATA0
- `dcsr.stepie`硬连线为0（单步执行期间无中断）
- `dcsr.stopcount`和`dcsr.stoptime`硬连线为1（调试模式下无计数器或内部定时器增量）
- `dcsr.mprven`硬连线为0
- `dcsr.prv`仅接受值3（M模式）和0（U模式），写入时采用四舍五入

有关核心侧调试模式寄存器的详细信息，请参见DCSR和DPC。

触发单元实现了四个精确指令地址匹配触发器。触发器可配置为陷入M模式及调试模式，这意味着M模式可利用触发器支持自托管硬件断点。`tcontrol.mte`和`tcontrol.mpte`字段的实现目的是防止当M模式触发器设定于M模式异常处理器时发生无限异常循环。

### 3.8.6. 自定义扩展

Hazard3实现了少量自定义扩展。所有扩展均为可选：仅当实例化处理器时相关功能标志置为1（参见第3.8.8节）时，才会包含自定义扩展。Hazard3始终是一个符合规范的RISC-V实现；当这些扩展禁用时，它同样是一个标准的RISC-V实现。

若启用任一扩展，则 MISA 寄存器中的 **x** 位将被置位，以指示存在非标准扩展。

### 3.8.6.1. Xh3irq: Hazard3 中断控制器

Xh3irq 最多支持 512 个外部中断，并提供多达 16 级的抢占优先级。其架构设计为建立于标准的 **mip.meip** 外部中断线之上，所有标准 RISC-V 中断行为依然适用。该扩展未新增指令，但添加了若干 CSR：

- **MEIEA**: 外部中断使能数组
- **MEIPA**: 外部中断挂起数组
- **MEIFA**: 外部中断强制数组
- **MEIPRA**: 外部中断优先级数组
- **MEINEXT**: 获取下一个外部中断
- **MEICONTEXT**: 外部中断上下文寄存器

Xh3irq 旨在支持以裸 C 函数形式实现的中断处理程序，调度由软件实现，抢占优先级逻辑由硬件实现。然而，具体的中断 ABI 由作为 **mip.meip** 外部中断处理程序安装的软件调度例程的实现决定。

#### 3.8.6.1.1. 数组CSR

RISC-V CSR 非常适合中断控制，因为它们与处理器紧密耦合，支持原子性的设置/清除访问，且可通过单条指令访问，无需先物化地址。然而，使用 CSR 处理大型位数组（例如中断使能）存在若干问题：

- CSR 地址空间有限
- CSR 无法通过间接寻址，因此难以迭代
- 由于附加的可变状态，使用 CSR 来索引其他 CSR 对中断处理程序而言存在问题

Xh3irq 采用 **array CSR** 惯用法，在单一 CSR 地址（如 MEIEA）处暴露大型位向量。该 CSR 的高 16 位作为数组的一个窗口，窗口通过相同 CSR 指令写入数据的最低有效位进行索引。

例如，以下汇编代码将 **0xa5a5** 写入中断使能数组的第 **47:32** 位，因窗口索引为 **0x2** 且窗口大小为 16 位：

```
li a0, 0xa5a50002
csrw RVCSR_MEIEA_OFFSET, a0
```

以下代码将中断挂起数组的第 **63:48** 位读取至寄存器 **a0**，因索引为 **0x3**，且一个 CSR set **0x0000** 不修改窗口内容：

```
csrrsi a0, RVCSR_MEIPA_OFFSET, 0x3
```

在 C 语言中设置任意 IRQ 使能的操作如下：

```
void enable_irq(uint irq) {
 uint index = irq / 16;
 uint32_t mask = 1u << (irq % 16);
 asm (
```

```

 "csrs 0xbe0, %0\n"
 : : "r" (index | (mask << 16))
);
}

```

在C语言中获取任意IRQ挂起标志的操作如下：

```

bool check_irq_pending(uint irq) {
 uint index = irq / 16;
 uint32_t csr_rdata;
 asm (
 "csrrs %0, 0xbe1, %1\n"
 : "=r" (csr_rdata)
 : "r" (index)
);
 csr_rdata >>= 16;
 return csr_rdata & (1u << (irq % 16));
}

```

SDK在[hardware\\_irq](#) API中实现了类似操作。

Hazard3 支持最多 512 个中断，即针对 5 位 CSR 立即数所有可能值的每个值分配一个 16 位窗口。

### 3.8.6.1.2. 使能、挂起与强制数组

MEIEA、MEIPA 和 MEIFA CSR 分别表示中断使能、挂起和强制数组。每个数组中包含对应每条系统级中断线的一位，系统总计有 52 条中断线。（有关系统 IRQ 号分配给外设的详情，请参见第 3.2 节。）

中断使能数组用于控制中断信号进入核心的权限。当 MEIEA 中某位被清零时，相关的中断信号将被忽略。当某位被置位时，断言的对应中断信号将会在安全且适当的时机引导核心进入 [meip](#) 向量。从该向量处，[meip](#) 处理程序保存被中断者上下文后跳转至正确的处理程序。

SDK 中[hardware\\_irq](#)库的[irq\\_set\\_enabled\(\)](#)函数提供了操作中断使能数组的便捷方法。

中断挂起数组显示系统级中断信号的当前状态。即使在 MEIEA 中相应位被清除，中断仍可在 MEIPA 中被检测到，即使该中断优先级不足以立即中断核心。此寄存器为只读：当相应中断源解除断言时，MEIPA 中的位会自动清除。例如，UART RX FIFO 中断在从 FIFO 读取数据后应自动清除。

中断强制数组使中断即使在相应系统级中断信号被解除断言时，仍显示为挂起状态。当 MEIFA 中的某一位被设置时，MEIPA 中对应位读为 1，如满足通常前提条件，将中断核心。

MEIFA 位在从 MEINEXT 采样相应中断时自动清除，无需向 MEINEXT.UPDATE 写入 1 位以清除中断强制位。这意味着设置 MEIFA 位应使中断被捕获 once。普通的 [csrw](#) 和 [csrw](#) 指令同样会清除 MEIFA。

六条备用中断线，编号为 46 至 51，在 SDK 中称为 SPAREIRQ\_IRQ\_0 至 SPAREIRQ\_IRQ\_5，故意未连接至系统级硬件。但它们仍然在中断控制器中完整实现，并在 MEIFA 中被设置为挂起时触发。例如，快速中断的上半部处理程序可以安排其较耗时的下半部以较低优先级执行，或者高优先级的上下文切换中断可能调度低优先级上下文切换，以清理堆栈上的中断帧。

### 3.8.6.1.3. 下一个中断寄存器

MEINEXT始终显示应处理的下一个中断，且考虑优先级顺序。仅当中断满足以下全部条件时，才会显示于MEINEXT：

1. 在MEIPA中处于挂起状态
2. 在MEIEA中被使能
3. 其优先级大于或等于MEICONTEXT.PPREEMPT

返回值为满足这三个条件的最高优先级中断的IRQ编号，左移两位。

当多个中断具有最高优先级时，将选择其中编号最低的中断作为决胜。

MEINEXT的最高有效位（MSB）被设置，表明没有符合条件的中断，且其余位在此情况下未定义。软件应反复读取MEINEXT，直至所有可用中断被处理完毕。`bltz`和`bgez`指令为测试寄存器最高有效位提供了方便手段。

上述规则3的目的是确保任何可能在被抢占中断帧中已执行的中断不会在当前帧中重新进入。若无此规则，同一中断处理程序多次执行可能因其他处理程序的抢占而交叉进行。程序员通常会对该情况感到惊讶。

`MEINEXT.UPDATE`为只写字段，用以指示硬件在该周期内将MEICONTEXT更新为关于MEINEXT中所示中断的信息。第3.8.6.1.5节详细说明了上下文寄存器的更新。

#### ！重要

随着中断信号的出现和消失，MEINEXT持续变化。对`MEINEXT.UPDATE`的写操作必须与读取MEINEXT中断索引的指令相同，以避免数据竞争。这可以通过`csrrw`或`csrrwi`指令实现。

### 3.8.6.1.4 中断优先级

中断优先级数组MEIPRA为每个中断实现了四位字段。在硬件中，数值更高（无符号）的MEIPRA值对应更高优先级，优先于低优先级中断。`irq_set_priority()` SDK函数采用相反约定，数值越低表示优先级越高。本节采用硬件编号。

MEIPRA中的中断优先级决定以下三项内容：

1. 中断源是否允许此时中断核心：其优先级必须大于或等于MEICONTEXT.PREEMPT
2. 中断源是否可以出现在MEINEXT：必须大于或等于MEICONTEXT.PPREEMPT。
3. 当MEINEXT存在多个候选时，该中断出现的顺序。

当MEICONTEXT被正确保存和恢复后，在中断处理程序之外，PREEMPT和PPREEMPT均为零；在中断处理程序内部，PREEMPT严格大于PPREEMPT。它们共同定义了可在无需推入或弹出中断堆栈帧的情况下处理的中断优先级范围。

在中断外操作中断优先级是安全的。写入优先级数组时无需禁用中断。在中断处理程序内部操作中断优先级须谨慎：硬件操作定义明确，但结果可能出乎意料。请注意以下情况：

1. 提升当前处理程序的优先级：若仍启用且待处理，将立即发生自我抢占。
2. 提升优先级低于MEICONTEXT.PPREEMPT的其他中断的优先级：该中断可能已在另一个因抢占您的处理程序而被中断的帧中处理中。提升优先级可能导致该处理程序在返回到仍处于执行中的原始帧之前于更高帧中执行，从而与自身执行交织。

如果正确保存和恢复MEICONTEXT，则PPREEMPT保证不会高于当前处理中断的优先级，因为它包含了抢占发生时PREEMPT的前一个值，而优先级低于

的中断无法中断内核。因此，对于上述情况 2，安全的近似处理方式为：切勿（以任何幅度）提升优先级低于当前运行处理程序的优先级。

如果必须提升优先级较低的中断的优先级，一种解决方案是将中断优先级的更新排队，并挂起分配给备用 IRQ 之一的最低优先级处理程序，以处理这些排队更新。

您可以通过在 MEIFA 中设置对应位来手动挂起该处理程序。该处理程序将在返回前台代码之前最后执行。这是安全的，因为最低优先级的中断按定义不能抢占任何其他中断。

### 3.8.6.1.5. 中断上下文管理

MEICONTEXT 寄存器具有两个功能：管理多个抢占中断堆栈帧间的核心抢占优先级，以及帮助软件跟踪当前正在执行的中断处理程序（如果存在）。

[MEICONTEXT.PREEMPT](#)、[MEICONTEXT.PPREEMPT](#)和[MEICONTEXT.PPPREEMPT](#)构成了三级抢占优先级堆栈：

- [PREEMPT](#) 设置中断核心的最低中断优先级
- [PPREEMPT](#)设置MEINEXT中出现的最低中断优先级：避免了可能已被抢占的中断处理程序的重复执行
- [PPPREEMPT](#)除保存/恢复PPREEMPT之外无其他硬件功能

在进入 [MIP.MEIP](#)向量时，硬件会与第3.8.4节所述的标准陷阱入口序列同步，原子性地对 [MEICONTEXT](#)进行以下更新：

1. 将当前 [MEICONTEXT.PPREEMPT](#) 的值保存至 [PPPREEMPT](#)
2. 将当前 [MEICONTEXT.PREEMPT](#) 的值保存至 [PPREEMPT](#)
3. 将引发该中断的 IRQ 优先级加一写入 [MEICONTEXT.PREEMPT](#)
4. 向 [MEICONTEXT.MRETEIRQ](#) 写入 1，以启用下一次 [mret](#) 的优先级恢复功能

标准陷阱入口序列包括清除 MSTATUS.MIE，故处理中断处理程序开始时中断处于禁用状态。

为实现抢占，[MIP.MEIP](#) 处理程序必须在上下文保存关键区段后重新启用中断。应保存 [MEICONTEXT](#)、[MSTATUS](#)、[MEPC](#) 以及调用者保存的通用寄存器。

任何非由 [MIP.MEIP](#) 触发的陷阱入口会清除 MRETEIRQ。陷阱退出 ([mret](#)) 亦会清除 MRETEIRQ。

当陷阱退出且 [MEICONTEXT.MRETEIRQ](#) 被设置时，会在标准陷阱退出序列同时，原子性地对 [MEICONTEXT](#) 进行以下更新：

1. 从 [MEICONTEXT.PPREEMPT](#) 恢复 [MEICONTEXT.PREEMPT](#)
2. 从 [MEICONTEXT.PPPREEMPT](#) 恢复 [MEICONTEXT.PPREEMPT](#)
3. 写入 0 至 [MEICONTEXT.PPPREEMPT](#)

MRETEIRQ 标志允许硬件将每个 [MIP.MEIP](#) 向量项与其对应的 [mret](#) 匹配。此操作平衡了 PREEMPT 优先级栈的入栈与出栈。当无抢占且中断处理程序中未引发异常时，可将 MRETEIRQ 保留在 [MEICONTEXT.MRETEIRQ](#) 寄存器中。否则，必须在进入外部中断向量时保存 [MEICONTEXT](#)，并在处理程序末尾的 [mret](#) 指令之前恢复。

保存和恢复期间必须禁用中断。

向 [MEINEXT.UPDATE](#) 写入 1 将按以下规则更新 [MEICONTEXT](#)：

1. 将 [MEINEXT.NOIRQ](#) 写入 [MEICONTEXT.NOIRQ](#)
2. 将 [MEINEXT.IRQ](#) (IRQ 号) 写入 [MEICONTEXT.IRQ](#)
3. 如果 [MEINEXT.NOIRQ](#) 是……
  - 清除：向 [MEICONTEXT.PREEMPT](#) 写入 [MEINEXT.IRQ](#) 的优先级加一

- 设置：向 MEICONTEXT.PREEMPT 写入 **0x10**（大于 MEIPRA 中任何中断优先级）

**MEICONTEXT.IRQ 和 NOIRQ 协助代码确定当前运行的是哪个中断处理程序。抢占当前中断的中断必须保存/恢复 MEICONTEXT，因此在处理程序期间检查这些字段是安全的。**

写入 MEINEXT.UPDATE 时对 MEICONTEXT.PREEMPT 的更新确保核心将被优先级高于即将进入中断的中断抢占。同样重要的是，它确保核心不会被优先级低或相等的中断抢占，包括即将进入处理程序的中断。

为避免应关注 Xh3irq 扩展的 MIP.MEIP 处理程序与可能不关注该扩展的 MTIP/MSIP 处理程序之间产生不当相互作用，最好避免后者抢占前者。

**MEICONTEXT.CLEARCS、MTIESAVE 和 MSIESAVE 支持在 MEIP 处理中断保存与恢复上下文期间，作为 MEICONTEXT CSR 访问的一部分，禁用并恢复定时器/软件中断使能。**

### 3.8.6.1.6. 最简处理程序示例

本示例演示一个最简的 **meip** 处理程序，该程序调度至由 C 函数组成的中断处理数组，且不启用抢占。在该情况下，MEIP RA 中配置的优先级仍决定多个中断同时断言时的中断进入顺序，但处理中断处理程序运行期间，直至其完成前，不处理其他中断。

```
#include "hardware/regs/rvcsr.h"

isr_riscv_machine_external_irq:
 // 进入 C ABI 函数前保存所有调用者保存寄存器和临时寄存器。
 // 注意，mstatus.mie 在中断入口时由硬件清除，且
 // 我们将保持该清除状态。
 addi sp, sp, -64
 sw ra, 0(sp)
 sw t0, 4(sp)
 sw t1, 8(sp)
 sw t2, 12(sp)
 sw a0, 16(sp)
 sw a1, 20(sp)
 sw a2, 24(sp)
 sw a3, 28(sp)
 sw a4, 32(sp)
 sw a5, 36(sp)
 sw a6, 40(sp)
 sw a7, 44(sp)
 sw t3, 48(sp)
 sw t4, 52(sp)
 sw t5, 56(sp)
 sw t6, 60(sp)

get_first_irq:
 // 从 meinext 采样当前最高优先级的活动 IRQ (左移 2 位)。// 不要设置 `update` 位，因为我们
 // 没有保存或恢复 meicontext —— // 这是允许的，只是意味着无法通过 meicontext 检查是否处于 IRQ。
 csrr a0, RVCSR_MEINEXT_OFFSET

 // 若当前优先级等级无活动中断请求，则最高有效位将被置位
 bltz a0, no_more_irqs

dispatch_irq:
 // 加载索引表项并跳转，无需边界检查，
 // 因为硬件不会返回不存在的IRQ。
 lui a1, %hi(__soft_vector_table)
 add a1, a1, a0
 lw a1, %lo(__soft_vector_table)(a1)
 jalr ra, a1

get_next_irq:
```

```

// 获取下一个最高优先级的中断请求
csrr a0, RVCCSR_MEINEXT_OFFSET
// 若当前优先级等级无活动中断请求，则最高有效位将被置位
bgez a0, dispatch_irq

no_more_irqs:
 // 恢复保存的上下文并从中断请求返回 lw ra, 0(sp)

 lw t0, 4(sp)
 lw t1, 8(sp)
 lw t2, 12(sp)
 lw a0, 16(sp)
 lw a1, 20(sp)
 lw a2, 24(sp)
 lw a3, 28(sp)
 lw a4, 32(sp)
 lw a5, 36(sp)
 lw a6, 40(sp)
 lw a7, 44(sp)
 lw t3, 48(sp)
 lw t4, 52(sp)
 lw t5, 56(sp)
 lw t6, 60(sp)
 addi sp, sp, 64
 mret

 // 中断处理函数指针数组
.section ".bss"
.p2align 2
.global __soft_vector_table
__soft_vector_table:
.space 52 * 4

```

由于处理程序在 `meinext` 上循环，直到没有更多中断待处理，因此多个中断可通过一次保存和恢复调用者保存寄存器及临时寄存器完成处理。

MEIPA 中每个 IRQ 的挂起状态在对应外设取消中断输出信号时清除。正确编写的中断处理程序应促使外设中断取消断言，因此每次从 `meinext` 读取均将返回新的中断。因为 `meinext` 始终返回优先级最高的有效中断，该循环按照优先级降序遍历有效中断。

执行软件寄存器保存/恢复的开销极小，因为保存/恢复例程受限于总线带宽，而非指令执行开销。这也使硬件更具灵活性，因为相同硬件能够支持多种中断 ABI。

### 3.8.6.2. Xh3pppm：M 模式 PMP 区域

该扩展新增了一个 M 模式 CSR，PMPCFGM0，允许在 M 模式下对 PMP 区域强制执行而无需锁定该区域。

当 PMP 用于非安全相关目的，如栈保护或外设访问的捕获与仿真时，该功能尤为实用。

### 3.8.6.3. Xh3power：Hazard3 电源管理

该扩展新增了一项 M 模式 CSR (MSLEEP) 及两条新的提示指令，`h3.block` 和 `h3.unblock`，位于 `slt` nop 兼容的自定义提示空间。

`msleepCSR` 控制处理器在 WFI 睡眠状态下的睡眠深度。默认情况下，WFI 被实现为

正常流水线停顿。通过适当配置msleep，处理器在睡眠时可关闭自身时钟，或通过简单的四阶段请求/确认握手，协商外部电源控制器以实现外部硬件的电源开启/关闭。  
这些选项可降低睡眠电流，但会增加唤醒延迟。

这些提示允许处理器在多处理器环境中进入睡眠状态，直到被其他处理器唤醒。它们基于标准 WFI 状态实现，因此与外部调试的交互方式相同，并且可利用 msleep 中的相同深度睡眠状态。

### 3.8.6.3.1. h3.block

进入 WFI 睡眠状态，直到收到解除阻塞信号或触发会导致 WFI 退出的中断。

若 mstatus.tw 被设置，则在低于 M 模式的特权级执行该指令将引发非法指令异常。

如果自上次 h3.block 以来已接收到解除阻塞信号，则该指令作为 nop 执行，处理器不会进入睡眠状态。从概念上讲，睡眠状态会立即跳过，因为对应的解除阻塞信号已被接收。

当邻近处理器（“邻近”的确切定义由实现者决定）执行 h3.unblock 指令，或因其他平台定义的其他原因时，将接收到解除阻塞信号。

该指令编码为 slt x0, x0, x0，属于自定义的兼容 nop 的提示编码空间。

示例 C 宏：

```
#define __h3_block() asm ("slt x0, x0, x0")
```

示例汇编宏：

```
.macro h3.block
 slt x0, x0, x0
.endm
```

### 3.8.6.3.2. h3.unblock

向系统中的其他处理器发出解除阻塞信号。例如，用于通知另一处理器工作队列目前非空。

若 mstatus.tw 被设置，则在低于 M 模式的特权级执行该指令将引发非法指令异常。

该指令编码为 slt x0, x0, x1，属于自定义的nop兼容提示编码空间。

示例 C 宏：

```
#define __h3_unblock() asm ("slt x0, x0, x1")
```

示例汇编宏：

```
.macro h3.unblock
 slt x0, x0, x1
.endm
```

### 3.8.6.4. Xh3.bextm: Hazard3 多位位提取指令

该扩展包含Zbs中“位提取”指令的多位版本，用于提取小范围连续位字段。

#### 3.8.6.4.1. h3.bextm

“位多提取”，为Zbs中 `bext` 指令的多位版本。先执行右移操作，随后对1至8位最低有效位进行掩码处理。

编码（R型）：

| 位     | 名称                       | 值       | 描述                                                                  |
|-------|--------------------------|---------|---------------------------------------------------------------------|
| 31:29 | <code>funct7[6:4]</code> | 0b000   | RES0                                                                |
| 28:26 | 大小                       | -       | 掩码中1的个数，值0→7表示1→8位。                                                 |
| 25    | <code>funct7[0]</code>   | 0b0     | RES0，因为与潜在的RV64版本中 <code>h3.bextm</code> 的 <code>shamt[5]</code> 对齐 |
| 24:20 | <code>rs2</code>         | -       | 源寄存器2（移位量）                                                          |
| 19:15 | <code>rs1</code>         | -       | 源寄存器1                                                               |
| 14:12 | <code>funct3</code>      | 0b000   | <code>h3.bextm</code>                                               |
| 11:7  | <code>rd</code>          | -       | 目标寄存器                                                               |
| 6:2   | <code>opc</code>         | 0b01011 | custom0 操作码                                                         |
| 1:0   | 大小                       | 0b11    | 32位指令                                                               |

示例C宏（使用GCC语句表达式）：

```
// nbits 必须为常量表达式
#define __h3_bextm(nbts, rs1, rs2) ({\
 uint32_t __h3_bextm_rd; \
 asm (" .insn r 0x0b, 0, %3, %0, %1, %2" \
 : "=r" (__h3_bextm_rd) \
 : "r" (rs1), "r" (rs2), "i" (((nbts) - 1) & 0x7) << 1) \
); \
 __h3_bextm_rd; \
})
```

示例汇编宏：

```
// rd = (rs1 >> rs2[4:0]) & ~(-1 << nbts)
.macro h3.bextm rd rs1 rs2 nbts
.if (\nbts < 1) || (\nbts > 8)
.err
.endif
#if NO_HAZARD3_CUSTOM
```

```

 srl \rd, \rs1, \rs2
 andi \rd, \rd, ((1 << \nbits) - 1)
#else
.insn r 0x0b, 0x0, (((\nbits - 1) & 0x7) << 1), \rd, \rs1, \rs2
#endif
.endm

```

### 3.8.6.4.2. h3.bextmi

**h3.bextm** 的立即数变体。

编码 (I型) :

| 位     | 名称        | 值       | 描述                     |
|-------|-----------|---------|------------------------|
| 31:29 | imm[11:9] | 0b000   | RES0                   |
| 28:26 | 大小        | -       | 掩码中1的个数，值0→7表示1→8位。    |
| 25    | imm[5]    | 0b0     | RES0，预留用于未来可能的 RV64 版本 |
| 24:20 | shamt     | -       | 移位量，范围 0 至 31          |
| 19:15 | rs1       | -       | 源寄存器1                  |
| 14:12 | funct3    | 0b100   | h3.bextmi              |
| 11:7  | rd        | -       | 目标寄存器                  |
| 6:2   | opc       | 0b01011 | custom0 操作码            |
| 1:0   | 大小        | 0b11    | 32位指令                  |

示例C宏（使用GCC语句表达式）：

```

// nbits 和 shamt 必须为常量表达式
#define __h3_bextmi(nbits, rs1, shamt) ({\
 uint32_t __h3_bextmi_rd; \
 asm (" .insn i 0x0b, 0x4, %0, %1, %2" \
 : "=r" (__h3_bextmi_rd) \
 : "r" (rs1), "i" (((nbits) - 1) & 0x7) << 6 | ((shamt) & 0x1f)) \
); \
 __h3_bextmi_rd; \
})

```

示例汇编宏：

```

// rd = (rs1 >> shamt) & ~(-1 << nbits)
.macro h3.bextmi rd rs1 shamt nbits
.if (\nbits < 1) || (\nbits > 8)
.err
.endif
.if (\shamt < 0) || (\shamt > 31)
.err
.endif
#if NO_HAZARD3_CUSTOM
 srli \rd, \rs1, \shamt
 andi \rd, \rd, ((1 << \nbits) - 1)
#else
 .insn i 0x0b, 0x4, \rd, \rs1, (\shamt & 0x1f) | (((\nbits - 1) & 0x7) << 6)

```

```
#endif
.endm
```

### 3.8.7. 指令周期计数

所有时序均假设总线行为完美（无下游总线阻塞）。

请参见第3.8.1.6节，指令行为概要。

#### 3.8.7.1. RV32I

| 指令                 | 周期        | 注意                          |
|--------------------|-----------|-----------------------------|
| <b>整数寄存器-寄存器</b>   |           |                             |
| add rd, rs1, rs2   | 1         |                             |
| sub rd, rs1, rs2   | 1         |                             |
| slt rd, rs1, rs2   | 1         |                             |
| sltu rd, rs1, rs2  | 1         |                             |
| and rd, rs1, rs2   | 1         |                             |
| or rd, rs1, rs2    | 1         |                             |
| xor rd, rs1, rs2   | 1         |                             |
| sll rd, rs1, rs2   | 1         |                             |
| srl rd, rs1, rs2   | 1         |                             |
| sra rd, rs1, rs2   | 1         |                             |
| <b>整数寄存器-立即数</b>   |           |                             |
| addi rd, rs1, imm  | 1         | nop 为伪指令，相当于 addi x0, x0, 0 |
| slti rd, rs1, imm  | 1         |                             |
| sltiu rd, rs1, imm | 1         |                             |
| andi rd, rs1, imm  | 1         |                             |
| ori rd, rs1, imm   | 1         |                             |
| xori rd, rs1, imm  | 1         |                             |
| slli rd, rs1, imm  | 1         |                             |
| srlti rd, rs1, imm | 1         |                             |
| srai rd, rs1, imm  | 1         |                             |
| <b>大立即数</b>        |           |                             |
| lui rd, imm        | 1         |                             |
| auipc rd, imm      | 1         |                             |
| <b>控制转移</b>        |           |                             |
| jal rd, 标签         | $2^{[1]}$ |                             |
| jalr rd, rs1, imm  | $2^{[1]}$ |                             |

| 指令                                | 周期                   | 注意                                   |
|-----------------------------------|----------------------|--------------------------------------|
| <code>beg rs1, rs2, label</code>  | 1 或 2 <sup>[1]</sup> | 预测正确为1， 错误为2。                        |
| <code>bne rs1, rs2, label</code>  | 1 或 2 <sup>[1]</sup> | 预测正确为1， 错误为2。                        |
| <code>blt rs1, rs2, label</code>  | 1 或 2 <sup>[1]</sup> | 预测正确为1， 错误为2。                        |
| <code>bge rs1, rs2, label</code>  | 1 或 2 <sup>[1]</sup> | 预测正确为1， 错误为2。                        |
| <code>bltu rs1, rs2, label</code> | 1 或 2 <sup>[1]</sup> | 预测正确为1， 错误为2。                        |
| <code>bgeu rs1, rs2, label</code> | 1 或 2 <sup>[1]</sup> | 预测正确为1， 错误为2。                        |
| 加载与存储                             |                      |                                      |
| <code>lw rd, imm(rs1)</code>      | 1 或 2                | 若下一指令独立，则为1， 若依赖，则为2。 <sup>[2]</sup> |
| <code>lh rd, imm(rs1)</code>      | 1 或 2                | 若下一指令独立，则为1， 若依赖，则为2。 <sup>[2]</sup> |
| <code>lhu rd, imm(rs1)</code>     | 1 或 2                | 若下一指令独立，则为1， 若依赖，则为2。 <sup>[2]</sup> |
| <code>lb rd, imm(rs1)</code>      | 1 或 2                | 若下一指令独立，则为1， 若依赖，则为2。 <sup>[2]</sup> |
| <code>lbu rd, imm(rs1)</code>     | 1 或 2                | 若下一指令独立，则为1， 若依赖，则为2。 <sup>[2]</sup> |
| <code>sw rs2, imm(rs1)</code>     | 1                    |                                      |
| <code>sh rs2, imm(rs1)</code>     | 1                    |                                      |
| <code>sb rs2, imm(rs1)</code>     | 1                    |                                      |

### 3.8.7.2. M 扩展

| 指令                               | 周期      | 注意      |
|----------------------------------|---------|---------|
| 32 × 32 → 32 乘法                  |         |         |
| <code>mul rd, rs1, rs2</code>    | 1       |         |
| 32 × 32 → 64 乘法，高位部分             |         |         |
| <code>mulh rd, rs1, rs2</code>   | 1       |         |
| <code>mulhsu rd, rs1, rs2</code> | 1       |         |
| <code>mulhu rd, rs1, rs2</code>  | 1       |         |
| 除法与余数                            |         |         |
| <code>div rd, rs1, rs2</code>    | 18 或 19 | 取决于符号修正 |
| <code>divu rd, rs1, rs2</code>   | 18      |         |
| <code>rem rd, rs1, rs2</code>    | 18 或 19 | 取决于符号修正 |
| <code>remu rd, rs1, rs2</code>   | 18      |         |

### 3.8.7.3. A 扩展

| 指令                               | 周期    | 注意                                                                                                              |
|----------------------------------|-------|-----------------------------------------------------------------------------------------------------------------|
| 加载保留／存储条件                        |       |                                                                                                                 |
| <code>lr.w rd, (rs1)</code>      | 1 或 2 | 如果下一条指令有依赖，则为 2 <sup>[2]</sup> ， 即 <code>lr.w</code> 、 <code>sc.w</code> 或 <code>amo*.w</code> 。 <sup>[3]</sup> |
| <code>sc.w rd, rs2, (rs1)</code> | 1 或 2 | 如果下一条指令有依赖，则为 2 <sup>[2]</sup> ， 即 <code>lr.w</code> 、 <code>sc.w</code> 或 <code>amo*.w</code> 。 <sup>[3]</sup> |

| 指令                                    | 周期 | 注意                                           |
|---------------------------------------|----|----------------------------------------------|
| 原子内存操作                                |    |                                              |
| <code>amoswap.w rd, rs2, (rs1)</code> | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amoadd.w rd, rs2, (rs1)</code>  | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amoxor.w rd, rs2, (rs1)</code>  | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amoand.w rd, rs2, (rs1)</code>  | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amoor.w rd, rs2, (rs1)</code>   | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amomin.w rd, rs2, (rs1)</code>  | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amamax.w rd, rs2, (rs1)</code>  | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amominu.w rd, rs2, (rs1)</code> | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |
| <code>amamaxu.w rd, rs2, (rs1)</code> | 4+ | 每次尝试 4 次。如果保留失效，可进行多次尝试。 <a href="#">[4]</a> |

### 3.8.7.4. C 扩展

所有 C 扩展的 16 位指令均为基础 RV32I 指令的别名。在 Hazard3 上，它们的执行效果与对应的 32 位指令完全一致。

C 扩展的一个结果是 32 位指令可以非自然对齐。这在顺序执行时无惩罚，但跳转到非 32 位自然对齐的指令时，会产生 1 个周期的惩罚，因为指令抓取被拆分为两个自然对齐的总线访问。

### 3.8.7.5. 特权指令（包括 Zicsr）

| 指令                               | 周期       | 注意                           |
|----------------------------------|----------|------------------------------|
| CSR 访问                           |          |                              |
| <code>csrrw rd, csr, rs1</code>  | 1        |                              |
| <code>csrrc rd, csr, rs1</code>  | 1        |                              |
| <code>csrrs rd, csr, rs1</code>  | 1        |                              |
| <code>csrrwi rd, csr, imm</code> | 1        |                              |
| <code>csrrci rd, csr, imm</code> | 1        |                              |
| <code>csrrsi rd, csr, imm</code> | 1        |                              |
| 陷阱与中断                            |          |                              |
| <code>ecall</code>               | 3        | 给定时间用于跳转至 <code>mtvec</code> |
| <code>ebreak</code>              | 3        | 给定时间用于跳转至 <code>mtvec</code> |
| <code>mret</code>                | $2^{11}$ |                              |
| <code>wfi</code>                 | 2+       | 始终停顿一个周期，无上限                 |

### 3.8.7.6. 位操作

| 指令                    | 周期 | 注意                               |
|-----------------------|----|----------------------------------|
| <b>Zba (地址生成)</b>     |    |                                  |
| sh1add rd, rs1, rs2   | 1  |                                  |
| sh2add rd, rs1, rs2   | 1  |                                  |
| sh3add rd, rs1, rs2   | 1  |                                  |
| <b>Zbb (基础位操作)</b>    |    |                                  |
| andn rd, rs1, rs2     | 1  |                                  |
| clz rd, rs1           | 1  |                                  |
| cpop rd, rs1          | 1  |                                  |
| ctz rd, rs1           | 1  |                                  |
| max rd, rs1, rs2      | 1  |                                  |
| maxu rd, rs1, rs2     | 1  |                                  |
| min rd, rs1, rs2      | 1  |                                  |
| minu rd, rs1, rs2     | 1  |                                  |
| orc.b rd, rs1         | 1  |                                  |
| orn rd, rs1, rs2      | 1  |                                  |
| rev8 rd, rs1          | 1  |                                  |
| rol rd, rs1, rs2      | 1  |                                  |
| ror rd, rs1, rs2      | 1  |                                  |
| rori rd, rs1, imm     | 1  |                                  |
| sext.b rd, rs1        | 1  |                                  |
| sext.h rd, rs1        | 1  |                                  |
| xnor rd, rs1, rs2     | 1  |                                  |
| zext.h rd, rs1        | 1  |                                  |
| zext.b rd, rs1        | 1  | zext.b 为伪操作，对应andi rd, rs1, 0xff |
| <b>Zbs (单个位操作)</b>    |    |                                  |
| bclr rd, rs1, rs2     | 1  |                                  |
| bclri rd, rs1, imm    | 1  |                                  |
| bext rd, rs1, rs2     | 1  |                                  |
| bexti rd, rs1, imm    | 1  |                                  |
| binv rd, rs1, rs2     | 1  |                                  |
| binvi rd, rs1, imm    | 1  |                                  |
| bset rd, rs1, rs2     | 1  |                                  |
| bseti rd, rs1, imm    | 1  |                                  |
| <b>Zbkb (加密基础位操作)</b> |    |                                  |
| pack rd, rs1, rs2     | 1  |                                  |
| packh rd, rs1, rs2    | 1  |                                  |

| 指令            | 周期 | 注意 |
|---------------|----|----|
| brev8 rd, rs1 | 1  |    |
| zip rd, rs1   | 1  |    |
| unzip rd, rs1 | 1  |    |

### 3.8.7.7. Zcb 扩展

与C扩展类似，本扩展包含常见32位指令的16位变体：

- RV32I基准ISA：`lbu`, `lh`, `luh`, `sb`, `sh`, `zext.b` (`andi`别名), `not` (`xori`别名)
- Zbb扩展：`sext.b`, `zext.h`, `sext.h`
- M扩展：`mul`

其性能与对应的32位指令完全一致。

### 3.8.7.8. Zcmp扩展

| 指令                                 | 周期                                                   | 注意                              |
|------------------------------------|------------------------------------------------------|---------------------------------|
| <code>cm.push rlist, -imm</code>   | $1 + n$                                              | $n$ 为 <code>rlist</code> 中寄存器数量 |
| <code>cm.pop rlist, imm</code>     | $1 + n$                                              | $n$ 为 <code>rlist</code> 中寄存器数量 |
| <code>cm.popret rlist, imm</code>  | $4 (n = 1)^{[5]} \text{ or } 2 + n (n \geq 2)^{[1]}$ | $n$ 为 <code>rlist</code> 中寄存器数量 |
| <code>cm.popretz rlist, imm</code> | $5 (n = 1)^{[5]} \text{ or } 3 + n (n \geq 2)^{[1]}$ | $n$ 为 <code>rlist</code> 中寄存器数量 |
| <code>cm.mva01s r1s', r2s'</code>  | 2                                                    |                                 |
| <code>cm.mvsa01 r1s', r2s'</code>  | 2                                                    |                                 |

### 3.8.7.9. 表格脚注

- [1] 对非32位对齐的32位指令进行跳转或分支需要额外一个周期，因为需要两个自然对齐的总线周期来获取目标指令。
- [2] 如果第2阶段的指令（例如 `add`）使用了第3阶段的数据（例如 `lw` 指令结果），则在这两者之间会插入一个1周期的气泡。加载数据 → 存储数据的依赖关系并非此类情况，因为数据在第3阶段被产生和使用。然而，加载数据 → 加载地址的情况则符合该例子，如同 `sc.w`。  
→ `beqz`.
- [3] AMO操作以总线上成对的独占读和独占写方式发出，最大访问速度为每次2周期，因总线不允许独占读/写的流水线化。如果写入阶段因全局监视器报告保留丢失而失败，指令会以每循环4个周期的速率循环，直到成功。如果全局监视器拒绝读取保留，指令将触发存储/原子操作异常（Store/AMO Fault exception），以防止无限循环。
- [4] 在 `lr.w/sc.w` 和紧随其后的 `lr.w/sc.w/amo*` 之间插入流水线气泡，因AHB5总线标准不允许流水线独占访问。无论如何，在 `lr.w` 和 `sc.w` 之间会插入停顿，以便根据 `lr.w` 的数据阶段及时更新本地监视器，从而抑制 `sc.w` 的地址阶段。
- [5] `cm.popret` 和 `cm.popretz` 的单寄存器版本所需周期与双寄存器版本相同，因内部存在对加载返回地址的加载-使用依赖。

### 3.8.7.10. 分支预测器

Hazard3 包含一个最简化分支预测器，用以加速紧密循环：

- 指令前端在单入口分支目标缓冲区（BTB）中记录最后一次采取的后向分支
- 如果再次遇到相同的分支，则预测其被采取
- 所有其他分支均预测为不采取
- 若核心执行但未采取预测为已采取的分支：
  - 核心将清除BTB
  - 该分支在下一次执行时预测为不采取

正确预测的分支以单周期执行：前端能够将两个非顺序的取指路径拼接，以使其表现为顺序执行。错误预测的分支会产生一个周期的惩罚，因为在分支执行时必须发出非顺序取指地址。请参见如下复制例程：

```
// a0 为目标指针
// a1 为源指针
// a2 为长度
copy_data:
 beqz a2, 2f
 add a2, a2, a1
1:
 lbu a3, (a0)
 sb a3, (a1)
 addi a0, a0, 1
 addi a1, a1, 1
 bltu a1, a2, 1b
2:
 返回
```

在稳态情况下，该操作每个循环执行5个周期：

- 加载操作占用一个周期。
- 存储操作占用一个周期：虽然依赖加载，但该依赖发生在第3阶段内，因此不会产生停顿。
- 每个 `add` 操作占用一个周期。
- 重复执行的后跳分支占用一个周期。

无分支预测时，吞吐率为每个循环6个周期。分支预测器提升吞吐率20%，并减少了因无效指令获取导致的能耗（对于嵌入式处理器，内存访问占指令能耗的最大比例）。

对于上述示例代码，拷贝10字节需要52个周期：

- 基本代价为每次迭代5个周期，迭代次数共计10次。
- 第一轮迭代末尾误预测且跳转的分支耗时一个周期。
- 最后一轮迭代末尾误预测且未跳转的分支耗时一个周期。

#### 3.8.7.10.1. 注意：延迟循环

当以下所有条件均满足时，分支预测器不会启动：

- 循环体仅包含单条16位指令（后接一条反复执行的后向分支）
- 循环体地址为32位对齐

- 指令取指端口无总线阻塞

这是因为分支预测器通过比较顺序取指计数器的位 31:2与 BTB 标签，实现查找。在此情况下，BTB 标签指向与循环入口相同的指令字。在上述情况下，顺序取指计数器从不实际包含循环入口地址，因为循环入口地址直接送往总线，顺序取指计数器提前递增至下一个地址。这在如下延迟循环中体现：

```
.p2align 2
delay_loop_bad_dont_copy_paste_this:
 addi a0, a0, -1
 bgez a0, delay_loop_bad_dont_copy_paste_this
```

根据第3.8.7.10节的描述，您可能预期该循环在稳定状态下每次迭代执行两个周期。实际情况是它在每次迭代执行三个周期，直到指令取指遇到阻塞，随后加速至每条指令两个周期，直到循环结束。

请通过在循环体中使用32位指令来避免此情况。强制循环体32位对齐以避免对齐惩罚。以下代码在稳定状态下如预期每次迭代执行两个周期：

```
.p2align 2 // 强制4字节对齐
delay_cycles:
.option push
.option norvc // 强制32位操作码
 addi a0, a0, -1
.option pop
 bgez a0, delay_cycles
```

### 3.8.8. 配置

Hazard3 使用 hazard3\_config.vh 头文件中给定的参数定制核心。这些数值在硅片流片前已设定，因此从用户角度来看它们是固定的。它们决定了处理器支持的指令、某些指令的面积与性能权衡，以及诸如PMP等核心外设的静态配置。RP2350对这些参数采用以下数值：

| 参数                 | 值 |
|--------------------|---|
| EXTENSION_A        | 1 |
| EXTENSION_C        | 1 |
| EXTENSION_M        | 1 |
| EXTENSION_ZBA      | 1 |
| EXTENSION_ZBB      | 1 |
| EXTENSION_ZBC      | 0 |
| EXTENSION_ZBS      | 1 |
| EXTENSION_ZCB      | 1 |
| EXTENSION_ZCMP     | 1 |
| EXTENSION_ZBKB     | 1 |
| EXTENSION_ZIFENCEI | 1 |
| EXTENSION_XH3BEXTM | 1 |
| EXTENSION_XH3IRQ   | 1 |

| 参数                   | 值                |
|----------------------|------------------|
| EXTENSION_XH3PPM     | 1                |
| EXTENSION_XH3POWER   | 1                |
| CSR_M_MANDATORY      | 1                |
| CSR_M_TRAP           | 1                |
| CSR_COUNTER          | 1                |
| U_MODE               | 1                |
| PMP_REGIONS          | 11               |
| PMP_GRAIN            | 3                |
| PMP_HARDWIRED        | 11'h700          |
| PMP_HARDWIRED_ADDR   | 参见第3.8.8.1节      |
| PMP_HARDWIRED_CFG    | 参见第3.8.8.1节      |
| DEBUG_SUPPORT        | 1                |
| BREAKPOINT_TRIGGERES | 4                |
| NUM_IRQS             | 52               |
| IRQ_PRIORITY_BITS    | 4                |
| IRQ_INPUT_BYPASS     | {NUM_IRQS{1'b1}} |
| MVENDORID_VAL        | 32'h00000493     |
| MIMPID_VAL           | 32'h86fc4e3f     |
| MCONFIGPTR_VAL       | 32'h0            |
| REDUCED_BYPASS       | 0                |
| MULDIV_UNROLL        | 2                |
| MUL_FAST             | 1                |
| MUL_FASTER           | 1                |
| MULH_FAST            | 1                |
| FAST_BRANCHCMP       | 1                |
| RESET_REGFILE        | 1                |
| BRANCH_PREDICTOR     | 1                |
| MTVEC_WMASK          | 32'hfffffffffd   |

### 3.8.8.1 硬连线PMP区域

RP2350在Hazard3中配置了八个动态PMP区域及三个静态PMP区域。静态区域为以下范围提供默认的用户态读写执行权限：

- 只读存储器： 0x00000000至 0xffffffff
- 外设： 0x40000000至 0x5fffffff
- SIO: 0xd0000000 至 0xffffffff

这些地址出现在 PMPADDR8、PMPADDR9 和 PMPADDR10 寄存器中。硬连线 PMP 地址寄存器的行为与动态寄存器相同，但它们忽略写入操作（遵循 WARL 规则）。这些权限

区域位于 PMPCFG2 中。

硬连线区域的功能类似于第 10.2.2 节中指定的 Cortex-M33 IDAU 地址映射中新增的豁免区域。

RP2350 对 AHB/APB 外设设置了默认的用户模式 (U-mode) 权限，因为这些外设预期通过 ACCESSCTRL (第 10.6 节) 进行分配。ACCESSCTRL 可利用总线结构中的现有地址解码器，逐个分配各外设，而 PMP 区域数量有限，因此在外设分配中作用有限。

同样，SIO 在安全/非安全总线归属内实施内部分区，映射至第 10.6.2 节所述的机器模式和用户模式。

动态区域 0 至 7 的优先级高于硬连线区域，因为 PMP 优先级按区域编号由低至高排序。

### 3.8.9. 控制与状态寄存器

控制和状态寄存器 (CSR) 是影响处理器行为的内部寄存器。它们是 hart本地的：

每个 hart拥有一份 CSR 的副本。在 RP2350 中，hart本地是 core本地的同义词。

应使用专用的 CSR 指令访问 CSR，如第 3.8.1.22 节所述。无法通过 load 或 store 指令访问 CSR。

RISC-V 特权规范对实施哪些 CSR 及其行为具有灵活性。本节专门记录了 Hazard3 上 CSR 的实际实现行为，未详尽列举所有平台的全部可能行为。

#### 重要

RISC-V 特权规范应作为编写运行于 Hazard3 软件的主要参考依据。

可移植的 RISC-V 软件不应依赖本节中描述的任何实现定义行为。

所有 CSR 均为 32 位，且 MXLEN 固定为 32 位。本节未列出的 CSR 地址未被实现。

访问未实现的 CSR 将引发非法指令异常 (`mcause=2`)。此处包含所有 S 模式控制状态寄存器 (CSRs)。

表367。  
RVCSR 寄存器列表

| 偏移量   | 名称            | 说明                                                |
|-------|---------------|---------------------------------------------------|
| 0x300 | MSTATUS       | 机器状态寄存器                                           |
| 0x301 | MISA          | ISA 扩展支持摘要                                        |
| 0x302 | MEDELEG       | 机器异常委托寄存器。未实现，因为不支持 S 状态。                         |
| 0x303 | MIDELEG       | 机器中断委托寄存器。未实现，因不支持 S 模式。                          |
| 0x304 | MIE           | 机器中断使能寄存器                                         |
| 0x305 | MTVEC         | 机器陷阱处理程序基址。                                       |
| 0x306 | MCOUNTEREN    | 计数器使能。控制来自用户态 (U 状态) 对计数器的访问。不可与 mcountinhibit混淆。 |
| 0x30a | MENVCFG       | 机器环境配置寄存器，低半字节                                    |
| 0x310 | MSTATUSH      | mstatus 的高半字节，硬连线为 0。                             |
| 0x31a | MENVCFGH      | 机器环境配置寄存器，高半字节                                    |
| 0x320 | MCOUNTINHIBIT | 用于 <code>mcycle/minstret</code> 的计数禁止寄存器          |
| 0x323 | MHPMEVENT3    | 扩展性能事件选择器，硬连线为 0。                                 |

| 偏移量   | 名称          | 说明                             |
|-------|-------------|--------------------------------|
| 0x324 | MHPMEVENT4  | 扩展性能事件选择器，硬连线为 0。              |
| 0x325 | MHPMEVENT5  | 扩展性能事件选择器，硬连线为 0。              |
| 0x326 | MHPMEVENT6  | 扩展性能事件选择器，硬连线为 0。              |
| 0x327 | MHPMEVENT7  | 扩展性能事件选择器，硬连线为 0。              |
| 0x328 | MHPMEVENT8  | 扩展性能事件选择器，硬连线为 0。              |
| 0x329 | MHPMEVENT9  | 扩展性能事件选择器，硬连线为 0。              |
| 0x32a | MHPMEVENT10 | 扩展性能事件选择器，硬连线为 0。              |
| 0x32b | MHPMEVENT11 | 扩展性能事件选择器，硬连线为 0。              |
| 0x32c | MHPMEVENT12 | 扩展性能事件选择器，硬连线为 0。              |
| 0x32d | MHPMEVENT13 | 扩展性能事件选择器，硬连线为 0。              |
| 0x32e | MHPMEVENT14 | 扩展性能事件选择器，硬连线为 0。              |
| 0x32f | MHPMEVENT15 | 扩展性能事件选择器，硬连线为 0。              |
| 0x330 | MHPMEVENT16 | 扩展性能事件选择器，硬连线为 0。              |
| 0x331 | MHPMEVENT17 | 扩展性能事件选择器，硬连线为 0。              |
| 0x332 | MHPMEVENT18 | 扩展性能事件选择器，硬连线为 0。              |
| 0x333 | MHPMEVENT19 | 扩展性能事件选择器，硬连线为 0。              |
| 0x334 | MHPMEVENT20 | 扩展性能事件选择器，硬连线为 0。              |
| 0x335 | MHPMEVENT21 | 扩展性能事件选择器，硬连线为 0。              |
| 0x336 | MHPMEVENT22 | 扩展性能事件选择器，硬连线为 0。              |
| 0x337 | MHPMEVENT23 | 扩展性能事件选择器，硬连线为 0。              |
| 0x338 | MHPMEVENT24 | 扩展性能事件选择器，硬连线为 0。              |
| 0x339 | MHPMEVENT25 | 扩展性能事件选择器，硬连线为 0。              |
| 0x33a | MHPMEVENT26 | 扩展性能事件选择器，硬连线为 0。              |
| 0x33b | MHPMEVENT27 | 扩展性能事件选择器，硬连线为 0。              |
| 0x33c | MHPMEVENT28 | 扩展性能事件选择器，硬连线为 0。              |
| 0x33d | MHPMEVENT29 | 扩展性能事件选择器，硬连线为 0。              |
| 0x33e | MHPMEVENT30 | 扩展性能事件选择器，硬连线为 0。              |
| 0x33f | MHPMEVENT31 | 扩展性能事件选择器，硬连线为 0。              |
| 0x340 | MSCRATCH    | 机器陷阱处理程序的暂存寄存器                 |
| 0x341 | MEPC        | 机器异常程序计数器                      |
| 0x342 | MCAUSE      | 机器陷阱原因进入陷阱时设置，用以指示陷阱的原因。软件可读写。 |
| 0x343 | MTVAL       | 机器错误地址或指令，硬连线为零。               |
| 0x344 | MIP         | 机器中断挂起                         |
| 0x3a0 | PMPCFG0     | 物理内存保护配置，涵盖区域0至3               |

| 偏移量   | 名称            | 说明                                     |
|-------|---------------|----------------------------------------|
| 0x3a1 | PMPCFG1       | 物理内存保护配置，涵盖区域4至7                       |
| 0x3a2 | PMPCFG2       | 物理内存保护配置，涵盖区域8至11                      |
| 0x3a3 | PMPCFG3       | 物理内存保护配置，涵盖区域12至15                     |
| 0x3b0 | PMPADDR0      | 区域0的物理内存保护地址                           |
| 0x3b1 | PMPADDR1      | 区域1的物理内存保护地址                           |
| 0x3b2 | PMPADDR2      | 区域2的物理内存保护地址                           |
| 0x3b3 | PMPADDR3      | 区域3的物理内存保护地址                           |
| 0x3b4 | PMPADDR4      | 区域4的物理内存保护地址                           |
| 0x3b5 | PMPADDR5      | 区域5的物理内存保护地址                           |
| 0x3b6 | PMPADDR6      | 第6区域的物理内存保护地址                          |
| 0x3b7 | PMPADDR7      | 第7区域的物理内存保护地址                          |
| 0x3b8 | PMPADDR8      | 第8区域的物理内存保护地址                          |
| 0x3b9 | PMPADDR9      | 第9区域的物理内存保护地址                          |
| 0x3ba | PMPADDR10     | 第10区域的物理内存保护地址                         |
| 0x3bb | PMPADDR11     | 第11区域的物理内存保护地址                         |
| 0x3bc | PMPADDR12     | 第12区域的物理内存保护地址                         |
| 0x3bd | PMPADDR13     | 第13区域的物理内存保护地址                         |
| 0x3be | PMPADDR14     | 第14区域的物理内存保护地址                         |
| 0x3bf | PMPADDR15     | 第15区域的物理内存保护地址                         |
| 0x7a0 | TSELECT       | 通过 <code>tdata1/tdata2</code> 配置的触发器选择 |
| 0x7a1 | TDATA1        | 触发器配置数据1                               |
| 0x7a2 | TDATA2        | 触发配置数据 2                               |
| 0x7b0 | DCSR          | 调试控制与状态寄存器（仅限调试模式）                     |
| 0x7b1 | DPC           | 调试程序计数器（仅限调试模式）                        |
| 0xb00 | MCYCLE        | 机器模式周期计数器，低半部分                         |
| 0xb02 | MINSTRET      | 机器模式已完成指令计数器，低半部分                      |
| 0xb03 | MHPMCOUNTER3  | 扩展性能计数器，硬连线为 0。                        |
| 0xb04 | MHPMCOUNTER4  | 扩展性能计数器，硬连线为 0。                        |
| 0xb05 | MHPMCOUNTER5  | 扩展性能计数器，硬连线为 0。                        |
| 0xb06 | MHPMCOUNTER6  | 扩展性能计数器，硬连线为 0。                        |
| 0xb07 | MHPMCOUNTER7  | 扩展性能计数器，硬连线为 0。                        |
| 0xb08 | MHPMCOUNTER8  | 扩展性能计数器，硬连线为 0。                        |
| 0xb09 | MHPMCOUNTER9  | 扩展性能计数器，硬连线为 0。                        |
| 0xb0a | MHPMCOUNTER10 | 扩展性能计数器，硬连线为 0。                        |

| 偏移量   | 名称             | 说明                |
|-------|----------------|-------------------|
| 0xb0b | MHPMCOUNTER11  | 扩展性能计数器，硬连线为 0。   |
| 0xb0c | MHPMCOUNTER12  | 扩展性能计数器，硬连线为 0。   |
| 0xb0d | MHPMCOUNTER13  | 扩展性能计数器，硬连线为 0。   |
| 0xb0e | MHPMCOUNTER14  | 扩展性能计数器，硬连线为 0。   |
| 0xb0f | MHPMCOUNTER15  | 扩展性能计数器，硬连线为 0。   |
| 0xb10 | MHPMCOUNTER16  | 扩展性能计数器，硬连线为 0。   |
| 0xb11 | MHPMCOUNTER17  | 扩展性能计数器，硬连线为 0。   |
| 0xb12 | MHPMCOUNTER18  | 扩展性能计数器，硬连线为 0。   |
| 0xb13 | MHPMCOUNTER19  | 扩展性能计数器，硬连线为 0。   |
| 0xb14 | MHPMCOUNTER20  | 扩展性能计数器，硬连线为 0。   |
| 0xb15 | MHPMCOUNTER21  | 扩展性能计数器，硬连线为 0。   |
| 0xb16 | MHPMCOUNTER22  | 扩展性能计数器，硬连线为 0。   |
| 0xb17 | MHPMCOUNTER23  | 扩展性能计数器，硬连线为 0。   |
| 0xb18 | MHPMCOUNTER24  | 扩展性能计数器，硬连线为 0。   |
| 0xb19 | MHPMCOUNTER25  | 扩展性能计数器，硬连线为 0。   |
| 0xb1a | MHPMCOUNTER26  | 扩展性能计数器，硬连线为 0。   |
| 0xb1b | MHPMCOUNTER27  | 扩展性能计数器，硬连线为 0。   |
| 0xb1c | MHPMCOUNTER28  | 扩展性能计数器，硬连线为 0。   |
| 0xb1d | MHPMCOUNTER29  | 扩展性能计数器，硬连线为 0。   |
| 0xb1e | MHPMCOUNTER30  | 扩展性能计数器，硬连线为 0。   |
| 0xb1f | MHPMCOUNTER31  | 扩展性能计数器，硬连线为 0。   |
| 0xb80 | MCYCLEH        | 机器模式周期计数器，高半部分    |
| 0xb82 | MINSTRETH      | 机器模式已完成指令计数器，低半部分 |
| 0xb83 | MHPMCOUNTER3H  | 扩展性能计数器，硬连线为 0。   |
| 0xb84 | MHPMCOUNTER4H  | 扩展性能计数器，硬连线为 0。   |
| 0xb85 | MHPMCOUNTER5H  | 扩展性能计数器，硬连线为 0。   |
| 0xb86 | MHPMCOUNTER6H  | 扩展性能计数器，硬连线为 0。   |
| 0xb87 | MHPMCOUNTER7H  | 扩展性能计数器，硬连线为 0。   |
| 0xb88 | MHPMCOUNTER8H  | 扩展性能计数器，硬连线为 0。   |
| 0xb89 | MHPMCOUNTER9H  | 扩展性能计数器，硬连线为 0。   |
| 0xb8a | MHPMCOUNTER10H | 扩展性能计数器，硬连线为 0。   |
| 0xb8b | MHPMCOUNTER11H | 扩展性能计数器，硬连线为 0。   |
| 0xb8c | MHPMCOUNTER12H | 扩展性能计数器，硬连线为 0。   |
| 0xb8d | MHPMCOUNTER13H | 扩展性能计数器，硬连线为 0。   |
| 0xb8e | MHPMCOUNTER14H | 扩展性能计数器，硬连线为 0。   |
| 0xb8f | MHPMCOUNTER15H | 扩展性能计数器，硬连线为 0。   |

| 偏移量   | 名称             | 说明                                                     |
|-------|----------------|--------------------------------------------------------|
| 0xb90 | MHPMCOUNTER16H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb91 | MHPMCOUNTER17H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb92 | MHPMCOUNTER18H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb93 | MHPMCOUNTER19H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb94 | MHPMCOUNTER20H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb95 | MHPMCOUNTER21H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb96 | MHPMCOUNTER22H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb97 | MHPMCOUNTER23H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb98 | MHPMCOUNTER24H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb99 | MHPMCOUNTER25H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb9a | MHPMCOUNTER26H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb9b | MHPMCOUNTER27H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb9c | MHPMCOUNTER28H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb9d | MHPMCOUNTER29H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb9e | MHPMCOUNTER30H | 扩展性能计数器，硬连线为 0。                                        |
| 0xb9f | MHPMCOUNTER31H | 扩展性能计数器，硬连线为 0。                                        |
| 0xbd0 | PMPCFGM0       | 将 PMP 区域设置为 M 模式，且不锁定                                  |
| 0xbe0 | MEIEA          | 外部中断使能数组                                               |
| 0xbe1 | MEIPA          | 外部中断挂起数组                                               |
| 0xbe2 | MEIFA          | 外部中断强制数组                                               |
| 0xbe3 | MEIPRA         | 外部中断优先级数组                                              |
| 0xbe4 | MEINEXT        | 获取下一个外部中断                                              |
| 0xbe5 | MEICONTEXT     | 外部中断上下文寄存器                                             |
| 0xbf0 | MSLEEP         | M 模式休眠控制寄存器                                            |
| 0xbff | DMDATA0        | 调试模块 DATA0 访问寄存器（仅限调试模式）                               |
| 0xc00 | 周期             | 只读用户模式别名，表示 mcycle, mcounteren.cy 设置时可访问               |
| 0xc02 | INSTRET        | 只读用户模式别名，表示 minstret, mcounteren.ir 设置时可访问             |
| 0xc80 | 周期高位           | 只读用户模式别名，表示 mcycleh, mcounteren.cy 设置时可访问              |
| 0xc82 | INSTRETH       | 只读用户模式别名，表示 minstreth, mcounteren.ir 设置时可访问            |
| 0xf11 | MVENDORID      | 厂商 ID                                                  |
| 0xf12 | MARCHID        | 架构标识符 (Hazard3)                                        |
| 0xf13 | MIMPID         | 实现标识。在 RP2350 上，该值为 0x86fc4e3f，表示 Hazard3 版本 v1.0-rc1。 |

| 偏移量   | 名称         | 说明                |
|-------|------------|-------------------|
| 0xf14 | MHARTID    | 硬件线程标识符           |
| 0xf15 | MCONFIGPTR | 指向配置数据结构的指针（固定为0） |

## RVCSR: MSTATUS寄存器

偏移: 0x300

### 描述

机器状态寄存器

表368。MSTATUS 寄存器

| 位     | 描述                                                                           | 类型 | 复位  |
|-------|------------------------------------------------------------------------------|----|-----|
| 31:22 | 保留。                                                                          | -  | -   |
| 21    | <b>TW:</b> 超时等待。当该位为1时，尝试在用户模式执行WFI指令将立即导致非法指令异常。                            | 读写 | 0x0 |
| 20:18 | 保留。                                                                          | -  | -   |
| 17    | <b>MPRV:</b> 修改权限。若为1，加载和存储操作将视当前权限级别为 mpp。这包括物理内存保护检查，以及系统总线上的特权级别和加载/存储地址。 | 读写 | 0x0 |
| 16:13 | 保留。                                                                          | -  | -   |
| 12:11 | <b>MPP:</b> 先前的特权级别。可存储值3（M模式）或0（U模式）。如果写入其他值，硬件将舍入到最近的支持模式。                 | 读写 | 0x3 |
| 10:8  | 保留。                                                                          | -  | -   |
| 7     | <b>MPIE:</b> 先前的中断使能。可读写。在陷阱进入时设置为mstatus.mie的当前值。在陷阱返回时设置为1。                | 读写 | 0x0 |
| 6:4   | 保留。                                                                          | -  | -   |
| 3     | <b>MIE:</b> 中断使能。可读写。在陷阱进入时设置为0。在陷阱返回时设置为mstatus.mie的当前值。                    | 读写 | 0x0 |
| 2:0   | 保留。                                                                          | -  | -   |

## RVCSR: MISA寄存器

偏移: 0x301

### 描述

ISA扩展支持摘要

在RP2350上，Hazard3 的完整 -march字符串为：rv32imac\_zicsr\_zifencei\_zba\_zbb\_zbs\_zbkb\_zca\_zcb\_zcmp

注意Zca在此情况下等同于C扩展；支持所有与32位非浮点处理器相关的RISC-V C扩展指令。在不支持Zc扩展的旧工具链中，适用的-march字符串为：rv32imac\_zicsr\_zifencei\_zba\_zbb\_zbs\_zbkb

此外，配置了以下自定义扩展：Xh3bm、Xh3power、Xh3irq、Xh3pppm。

表369. MISA 寄存器

| 位     | 描述                             | 类型 | 复位  |
|-------|--------------------------------|----|-----|
| 31:30 | <b>MXL:</b> 值为0x1表示这是一个32位处理器。 | 只读 | 0x1 |
| 29:24 | 保留。                            | -  | -   |

| 位     | 描述                                                                                                                              | 类型 | 复位  |
|-------|---------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 23    | <b>X:</b> 值为1表示存在非标准扩展。 (Xh3b位操作及自定义睡眠和中断控制CSR)                                                                                 | 只读 | 0x1 |
| 22    | 保留。                                                                                                                             | -  | -   |
| 21    | <b>V:</b> 向量扩展（未实现）。                                                                                                            | 只读 | 0x0 |
| 20    | <b>U:</b> 值为1表示已实现U模式。                                                                                                          | 只读 | 0x1 |
| 19    | 保留。                                                                                                                             | -  | -   |
| 18    | <b>S:</b> 主管扩展（未实现）。                                                                                                            | 只读 | 0x0 |
| 17    | 保留。                                                                                                                             | -  | -   |
| 16    | <b>Q:</b> 四倍精度浮点扩展（未实现）。                                                                                                        | 只读 | 0x0 |
| 15:13 | 保留。                                                                                                                             | -  | -   |
| 12    | <b>M:</b> 值为1表示已实现M扩展（整数乘除法）。                                                                                                   | 只读 | 0x1 |
| 11:9  | 保留。                                                                                                                             | -  | -   |
| 8     | <b>I:</b> 值为1表示已实现RVI基础ISA（区别于RVE）。                                                                                             | 只读 | 0x1 |
| 7     | <b>H:</b> 管理程序扩展（尚未实现，尽管我同意若用于微控制器将非常酷）。                                                                                        | 只读 | 0x0 |
| 6     | 保留。                                                                                                                             | -  | -   |
| 5     | <b>F:</b> 单精度浮点扩展（尚未实现）。                                                                                                        | 只读 | 0x0 |
| 4     | <b>E:</b> RV32E/64E基础ISA（尚未实现）。                                                                                                 | 只读 | 0x0 |
| 3     | <b>D:</b> 双精度浮点扩展（尚未实现）。                                                                                                        | 只读 | 0x0 |
| 2     | <b>C:</b> 值为1表示已实现C扩展（压缩指令）。                                                                                                    | 只读 | 0x1 |
| 1     | <b>B:</b> 值为1表示已实现B扩展（位操作）。 B是Zba、Zbb和Zbs的组合。<br><br>Hazard3实现了所有这些扩展，但在Hazard3本版本流片时，B定义为ZbaZbbZbs尚未存在。<br>该位当时保留为0。因此该位读取值为0。 | 只读 | 0x0 |
| 0     | <b>A:</b> 值为1表示已实现A扩展（原子操作）。                                                                                                    | 只读 | 0x1 |

## RVCSR: MEDELEG寄存器

偏移: 0x302

表370. MEDELEG  
寄存器

| 位    | 描述                     | 类型 | 复位 |
|------|------------------------|----|----|
| 31:0 | 机器异常委托寄存器。未实现，因为不支持S态。 | 读写 | -  |

## RVCSR: MIDELEG寄存器

偏移: 0x303

表371. MDELEG  
寄存器

| 位    | 描述                     | 类型 | 复位 |
|------|------------------------|----|----|
| 31:0 | 机器中断委托寄存器。未实现，因为不支持S态。 | 读写 | -  |

## RVCSR: MIE寄存器

偏移: 0x304

### 描述

机器中断使能寄存器

表372. MIE  
寄存器

| 位     | 描述                                                                                                                                                                                              | 类型 | 复位  |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:12 | 保留。                                                                                                                                                                                             | -  | -   |
| 11    | <b>MEIE:</b> 外部中断使能。当 <code>mie.meie</code> 、 <code>mip.meip</code> 及 <code>mstatus mie</code> 均为1时，处理器跳转至外部中断向量。<br><br>Hazard3设有内部寄存器，可单独过滤外部中断（参见 <code>meiea</code> ），但此标准控制可用于一次性屏蔽所有外部中断。 | 读写 | 0x0 |
| 10:8  | 保留。                                                                                                                                                                                             | -  | -   |
| 7     | <b>MTIE:</b> 定时器中断使能。当 <code>mie.mtie</code> 、 <code>mip.mtip</code> 及 <code>mstatus mie</code> 均为1时，处理器跳转至定时器中断向量，除非此时软件或外部中断请求亦处于挂起且使能状态。                                                     | 读写 | 0x0 |
| 6:4   | 保留。                                                                                                                                                                                             | -  | -   |
| 3     | <b>MSIE:</b> 软件中断使能。当 <code>mie.msie</code> 、 <code>mip.msip</code> 和 <code>mstatus mie</code> 均为1时，处理器将跳转至软件中断向量，除非此时同时有外部中断请求处于待处理且已使能状态。                                                     | 读写 | 0x0 |
| 2:0   | 保留。                                                                                                                                                                                             | -  | -   |

## RVCSR: MTVEC寄存器

偏移: 0x305

### 描述

机器陷阱处理程序地址。

表373. MTVEC  
寄存器

| 位    | 描述                                                                                                                                                         | 类型 | 复位         |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------------|
| 31:2 | <b>BASE:</b> 陷阱向量地址的高30位（最低2位隐含为0）。<br>启用向量化时，必须64字节对齐。否则，必须4字节对齐。                                                                                         | 读写 | 0x00001fff |
| 1:0  | <b>MODE:</b> 若为0（直接模式），所有陷阱均将程序计数器设为陷阱向量基址。若为1（向量模式），异常将程序计数器设为陷阱向量基址，中断将程序计数器设为中断原因号乘以4（3=软件中断，7=定时器中断，11=外部中断）。<br><br>最高位硬连线为0，试图将mode设置为2或3时，结果分别为0或1。 | 读写 | 0x0        |
|      | 枚举值：                                                                                                                                                       |    |            |
|      | 0x0 → DIRECT: mtvec 的直接入口                                                                                                                                  |    |            |
|      | 0x1 → VECTORED: 以 mtvec 为起点的 16 项跳转表的向量入口                                                                                                                  |    |            |

## RVCSR: MCOUNTEREN 寄存器

偏移: 0x306

### 描述

计数器使能。控制来自用户态（U态）对计数器的访问。不可与mcountinhibit混淆。

表 374.  
MCOUNTEREN  
寄存器

| 位    | 描述                                                                                                                | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:3 | 保留。                                                                                                               | -  | -   |
| 2    | <b>IR</b> : 若值为 1, U 模式允许访问 <code>instret</code> / <code>instreth</code> 指令退休计数器 CSR。否则，U 模式对这些 CSR 的访问将触发异常。     | 读写 | 0x0 |
| 1    | <b>TM</b> : 无硬件效应，因为未实现 <code>time</code> / <code>timeh</code> CSR。<br>但该字段仍然存在，供 M 模式软件用于跟踪是否应仿真 U 模式尝试访问这些 CSR。 | 读写 | 0x0 |
| 0    | <b>CY</b> : 若值为 1, U 模式允许访问 <code>cycle</code> / <code>cycleh</code> 周期计数器 CSR。<br>否则，U 模式对这些 CSR 的访问将触发异常。       | 读写 | 0x0 |

## RVCSR: MENVCFG 寄存器

偏移: 0x30a

### 描述

机器环境配置寄存器，低半字节

表375. MENVCFG  
寄存器

| 位    | 描述                                                                                                                                             | 类型 | 复位  |
|------|------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:1 | 保留。                                                                                                                                            | -  | -   |
| 0    | <b>FIOM</b> : 设置后，在低于M模式权限的模式中，指定IO内存访问有序的fence 指令，也将导致主存访问的有序。<br><br>FIOM在Hazard3中硬连线为零，因其不支持S模式，且fence指令执行时均作为NOP（除 <code>fence.i</code> 外） | 只读 | 0x0 |

## RVCSR: MSTATUSH 寄存器

偏移: 0x310

表376. MSTATUSH  
寄存器

| 位    | 描述                  | 类型 | 复位         |
|------|---------------------|----|------------|
| 31:0 | mstatus的高半字节，硬连线为0。 | 只读 | 0x00000000 |

## RVCSR: MENVCFGH 寄存器

偏移: 0x31a

### 描述

机器环境配置寄存器，高半字节

该寄存器完全保留，因Hazard3未实现相关扩展。其实现为硬连线零。

表377.  
MENVCFGH寄存器

| 位    | 描述  | 类型 | 复位 |
|------|-----|----|----|
| 31:0 | 保留。 | -  | -  |

## RVCSR: MCOUNTINHIBIT寄存器

偏移: 0x320

### 描述

用于 `mcycle/minstret` 的计数禁止寄存器

表378.  
MCOUNTINHIBIT  
T寄存器

| 位    | 描述                                                                           | 类型 | 复位  |
|------|------------------------------------------------------------------------------|----|-----|
| 31:3 | 保留。                                                                          | -  | -   |
| 2    | <b>IR</b> : 禁止计数 <code>minstret</code> 和 <code>minstreth</code> 寄存器。默认设置以节能。 | 读写 | 0x1 |
| 1    | 保留。                                                                          | -  | -   |
| 0    | <b>CY</b> : 禁止计数 <code>mcycle</code> 和 <code>mcycleh</code> 寄存器。默认设置以节能。     | 读写 | 0x1 |

## RVCSR: MHPMEVENT3、MHPMEVENT4、...、MHPMEVENT30、MHPMEVENT31 寄存器

偏移量: 0x323, 0x324, ..., 0x33e, 0x33f

表379。  
MHPMEVENT3、  
MHPMEVENT4、...  
、MHPMEVENT30  
、MHPMEVENT31  
寄存器

## RVCSR: MSCRATCH寄存器

偏移量: 0x340

表380。  
MSCRATCH寄存器

| 位    | 描述                 | 类型 | 复位         |
|------|--------------------|----|------------|
| 31:0 | 扩展性能事件选择器, 硬连线为 0。 | 只读 | 0x00000000 |

## RVCSR: MEPC寄存器

偏移: 0x341

表381. MEPC  
寄存器

| 位    | 描述                                                                                                     | 类型 | 复位         |
|------|--------------------------------------------------------------------------------------------------------|----|------------|
| 31:2 | 机器异常程序计数器。<br><br>进入陷阱时, 程序计数器的当前值记录于此处。执行 <code>mret</code> 时, 处理器跳转至 <code>mepc</code> 。该寄存器亦可被软件读写。 | 读写 | 0x00000000 |
| 1:0  | 保留。                                                                                                    | -  | -          |

## RVCSR: MCAUSE 寄存器

偏移: 0x342

**描述**

机器陷阱原因进入陷阱时设置，用以指示陷阱的原因。软件可读写。

表 382. MCAUSE  
寄存器

| 位    | 描述                                                                                                                            | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31   | 中断: 若值为1，则陷阱由中断引起。若值为0，则由异常引起。                                                                                                | 读写 | 0x0 |
| 30:4 | 保留。                                                                                                                           | -  | -   |
| 3:0  | 代码: 若 <code>interrupt</code> 被设置, <code>code</code> 表示引发陷阱的mip位索引 (3=软中断, 7=定时器中断, 11=外部中断) ; 否则, <code>code</code> 根据异常原因设定。 | 读写 | 0x0 |
|      | 枚举值:                                                                                                                          |    |     |
|      | 0x0 → INSTR_ALIGN: 指令取指地址未对齐。由于启用了C扩展, 此项在RP2350上永不触发。                                                                        |    |     |
|      | 0x1 → INSTR_FAULT: 指令访问错误。指令获取未通过PMP检查, 或遇到下游总线故障, 随后越过不可推测点。                                                                 |    |     |
|      | 0x2 → ILLEGAL_INSTR: 执行了非法指令 (包括非法CSR访问)。                                                                                     |    |     |
|      | 0x3 → BREAKPOINT: 断点。当相关dcsr.ebreak位清除时执行ebreak指令。                                                                            |    |     |
|      | 0x4 → LOAD_ALIGN: 加载地址未对齐。Hazard3要求所有访问均为自然对齐。                                                                                |    |     |
|      | 0x5 → LOAD_FAULT: 加载访问错误。加载未通过PMP检查, 或遇到下游总线故障。                                                                               |    |     |
|      | 0x6 → STORE_ALIGN: 存储/AMO地址未对齐。Hazard3要求所有访问均为自然对齐。                                                                           |    |     |
|      | 0x7 → STORE_FAULT: 存储/AMO访问错误。存储/AMO未通过PMP检查, 或遇到下游总线故障。若对不支持原子操作的区域尝试执行AMO (RP2350上除SRAM外的任何区域), 该标志亦将被设定。                   |    |     |
|      | 0x8 → U_ECALL: 来自U模式的环境调用。                                                                                                    |    |     |
|      | 0xb → M_ECALL: 来自M模式的环境调用。                                                                                                    |    |     |

**RVCSR: MTVAL 寄存器**

偏移: 0x343

表383. MTVAL  
寄存器

| 位    | 描述                | 类型 | 复位         |
|------|-------------------|----|------------|
| 31:0 | 机器错误地址或指令, 硬连线为零。 | 只读 | 0x00000000 |

**RVCSR: MIP 寄存器**

偏移: 0x344

**描述**

机器中断挂起

表384. MIP 寄存器

| 位     | 描述                                                                                                                                                                                                                                                                                                                              | 类型 | 复位  |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:12 | 保留。                                                                                                                                                                                                                                                                                                                             | -  | -   |
| 11    | <b>MEIP</b> : 外部中断挂起。当 <code>mie.meie</code> 、 <code>mip.meip</code> 及 <code>mstatus mie</code> 均为1时，处理器跳转至外部中断向量。<br><br>Hazard3 具有内部寄存器，可单独过滤哪些外部 IRQ 会出现在 <code>meip</code> 中。当 <code>meip</code> 为1时，表示至少有一个外部中断被断言（故在 <code>meipa</code> 中挂起）、在 <code>meiea</code> 中启用，且优先级大于或等于当前 <code>meicontext.preempt</code> 中的抢占级别。 | 只读 | 0x0 |
| 10:8  | 保留。                                                                                                                                                                                                                                                                                                                             | -  | -   |
| 7     | <b>MTIP</b> : 定时器中断挂起。当 <code>mie.mtie</code> 、 <code>mip.mtip</code> 及 <code>mstatus mie</code> 均为1时，处理器跳转至定时器中断向量，除非此时软件或外部中断请求亦处于挂起且使能状态。                                                                                                                                                                                    | 读写 | 0x0 |
| 6:4   | 保留。                                                                                                                                                                                                                                                                                                                             | -  | -   |
| 3     | <b>MSIP</b> : 软件中断挂起。当 <code>mie.msie</code> 、 <code>mip.msip</code> 和 <code>mstatus mie</code> 均为1时，处理器将跳转至软件中断向量，除非此时同时有外部中断请求处于待处理且已使能状态。                                                                                                                                                                                    | 读写 | 0x0 |
| 2:0   | 保留。                                                                                                                                                                                                                                                                                                                             | -  | -   |

## RVCSR: PMPCFG0 寄存器

偏移: 0x3a0

### 描述

物理内存保护配置，适用于区域0至区域3

表385. PMPCFG0 寄存器

| 位     | 描述                                                  | 类型 | 复位  |
|-------|-----------------------------------------------------|----|-----|
| 31    | <b>R3_L</b> : 锁定区域3，并应用于M模式和U模式。                    | 读写 | 0x0 |
| 30:29 | 保留。                                                 | -  | -   |
| 28:27 | <b>R3_A</b> : 区域3的地址匹配类型。写入不支持的值 (TOR) 将使该区域被禁用。    | 读写 | 0x0 |
|       | 枚举值：                                                |    |     |
|       | 0x0 → 关闭：禁用该区域                                      |    |     |
|       | 0x2 → NA4：自然对齐4字节                                   |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                     |    |     |
| 26    | <b>R3_R</b> : 区域3的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 25    | <b>R3_W</b> : 区域3的写入权限。                             | 读写 | 0x0 |
| 24    | <b>R3_X</b> : 区域3的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 23    | <b>R2_L</b> : 锁定区域2，并应用于M模式和U模式。                    | 读写 | 0x0 |
| 22:21 | 保留。                                                 | -  | -   |
| 20:19 | <b>R2_A</b> : 第2区域的地址匹配类型。写入不支持的值 (TOR) 将使该区域被禁用。   | 读写 | 0x0 |

| 位     | 描述                                                   | 类型 | 复位  |
|-------|------------------------------------------------------|----|-----|
|       | 枚举值：                                                 |    |     |
|       | 0x0 → 关闭：禁用该区域                                       |    |     |
|       | 0x2 → NA4：自然对齐4字节                                    |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                      |    |     |
| 18    | <b>R2_R</b> : 第2区域的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 17    | <b>R2_W</b> : 第2区域的写入权限。                             | 读写 | 0x0 |
| 16    | <b>R2_X</b> : 第2区域的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 15    | <b>R1_L</b> : 锁定第1区域，并应用于M模式及U模式。                    | 读写 | 0x0 |
| 14:13 | 保留。                                                  | -  | -   |
| 12:11 | <b>R1_A</b> : 第1区域的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。<br>◦ | 读写 | 0x0 |
|       | 枚举值：                                                 |    |     |
|       | 0x0 → 关闭：禁用该区域                                       |    |     |
|       | 0x2 → NA4：自然对齐4字节                                    |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                      |    |     |
| 10    | <b>R1_R</b> : 第1区域的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 9     | <b>R1_W</b> : 第1区域的写入权限。                             | 读写 | 0x0 |
| 8     | <b>R1_X</b> : 第1区域的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 7     | <b>R0_L</b> : 锁定第0区域，并应用于M模式及U模式。                    | 读写 | 0x0 |
| 6:5   | 保留。                                                  | -  | -   |
| 4:3   | <b>R0_A</b> : 第0区域的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。<br>◦ | 读写 | 0x0 |
|       | 枚举值：                                                 |    |     |
|       | 0x0 → 关闭：禁用该区域                                       |    |     |
|       | 0x2 → NA4：自然对齐4字节                                    |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                      |    |     |
| 2     | <b>R0_R</b> : 第0区域的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 1     | <b>R0_W</b> : 第0区域的写入权限。                             | 读写 | 0x0 |
| 0     | <b>R0_X</b> : 第0区域的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |

## RVCSR: PMPCFG1寄存器

偏移: 0x3a1

**描述**

区域4至区域7的物理内存保护配置

表386. PMPCFG1  
寄存器

| 位     | 描述                                                  | 类型 | 复位  |
|-------|-----------------------------------------------------|----|-----|
| 31    | <b>R7_L:</b> 锁定区域7，并将其应用于M模式及U模式。                   | 读写 | 0x0 |
| 30:29 | 保留。                                                 | -  | -   |
| 28:27 | <b>R7_A:</b> 区域7的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。       | 读写 | 0x0 |
|       | 枚举值：                                                |    |     |
|       | 0x0 → 关闭：禁用该区域                                      |    |     |
|       | 0x2 → NA4：自然对齐4字节                                   |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                     |    |     |
| 26    | <b>R7_R:</b> 区域7的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。  | 读写 | 0x0 |
| 25    | <b>R7_W:</b> 区域7的写入权限。                              | 读写 | 0x0 |
| 24    | <b>R7_X:</b> 区域7的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。  | 读写 | 0x0 |
| 23    | <b>R6_L:</b> 锁定区域6，并将其应用于M模式及U模式。                   | 读写 | 0x0 |
| 22:21 | 保留。                                                 | -  | -   |
| 20:19 | <b>R6_A:</b> 区域6的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。       | 读写 | 0x0 |
|       | 枚举值：                                                |    |     |
|       | 0x0 → 关闭：禁用该区域                                      |    |     |
|       | 0x2 → NA4：自然对齐4字节                                   |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                     |    |     |
| 18    | <b>R6_R:</b> 区域6的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。  | 读写 | 0x0 |
| 17    | <b>R6_W:</b> 区域6的写入权限。                              | 读写 | 0x0 |
| 16    | <b>R6_X:</b> 区域6的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。  | 读写 | 0x0 |
| 15    | <b>R5_L:</b> 锁定区域5，并将其应用于M模式及U模式。                   | 读写 | 0x0 |
| 14:13 | 保留。                                                 | -  | -   |
| 12:11 | <b>R5_A:</b> 第5区域的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。      | 读写 | 0x0 |
|       | 枚举值：                                                |    |     |
|       | 0x0 → 关闭：禁用该区域                                      |    |     |
|       | 0x2 → NA4：自然对齐4字节                                   |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                     |    |     |
| 10    | <b>R5_R:</b> 第5区域的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 读写 | 0x0 |
| 9     | <b>R5_W:</b> 第5区域的写入权限。                             | 读写 | 0x0 |

| 位   | 描述                                                                                                                                    | 类型 | 复位  |
|-----|---------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 8   | <b>R5_X:</b> 第5区域的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。                                                                                   | 读写 | 0x0 |
| 7   | <b>R4_L:</b> 锁定第4区域，并将其应用于M模式和U模式。                                                                                                    | 读写 | 0x0 |
| 6:5 | 保留。                                                                                                                                   | -  | -   |
| 4:3 | <b>R4_A:</b> 第4区域的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。<br>○<br>枚举值：<br>0x0 → 关闭：禁用该区域<br>0x2 → NA4：自然对齐4字节<br>0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB） | 读写 | 0x0 |
| 2   | <b>R4_R:</b> 第4区域的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。                                                                                   | 读写 | 0x0 |
| 1   | <b>R4_W:</b> 第4区域的写入权限。                                                                                                               | 读写 | 0x0 |
| 0   | <b>R4_X:</b> 第4区域的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。                                                                                   | 读写 | 0x0 |

## RVCSR：PMPCFG2寄存器

偏移：0x3a2

### 说明

第8至第11区域的物理内存保护配置

表387. PMPCFG2 寄存器

| 位     | 描述                                                                                                                                      | 类型 | 复位  |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31    | <b>R11_L:</b> 锁定第11区域，并将其应用于M模式和U模式。                                                                                                    | 只读 | 0x0 |
| 30:29 | 保留。                                                                                                                                     | -  | -   |
| 28:27 | <b>R11_A:</b> 第11区域的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。<br>○<br>枚举值：<br>0x0 → 关闭：禁用该区域<br>0x2 → NA4：自然对齐4字节<br>0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB） | 只读 | 0x0 |
| 26    | <b>R11_R:</b> 区域11的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。                                                                                    | 只读 | 0x0 |
| 25    | <b>R11_W:</b> 区域11的写入权限。                                                                                                                | 只读 | 0x0 |
| 24    | <b>R11_X:</b> 区域11的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。                                                                                    | 只读 | 0x0 |
| 23    | <b>R10_L:</b> 锁定区域10，且同时适用于M模式和U模式。                                                                                                     | 只读 | 0x0 |
| 22:21 | 保留。                                                                                                                                     | -  | -   |
| 20:19 | <b>R10_A:</b> 区域10的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。<br>○<br>枚举值：                                                                            | 只读 | 0x3 |

| 位     | 描述                                                    | 类型 | 复位  |
|-------|-------------------------------------------------------|----|-----|
|       | 0x0 → 关闭：禁用该区域                                        |    |     |
|       | 0x2 → NA4：自然对齐4字节                                     |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                       |    |     |
| 18    | <b>R10_R</b> : 区域10的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x1 |
| 17    | <b>R10_W</b> : 区域10的写入权限。                             | 只读 | 0x1 |
| 16    | <b>R10_X</b> : 区域10的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x1 |
| 15    | <b>R9_L</b> : 锁定区域9，且同时适用于M模式和U模式。                    | 只读 | 0x0 |
| 14:13 | 保留。                                                   | -  | -   |
| 12:11 | <b>R9_A</b> : 区域9的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。        | 只读 | 0x3 |
|       | 枚举值：                                                  |    |     |
|       | 0x0 → 关闭：禁用该区域                                        |    |     |
|       | 0x2 → NA4：自然对齐4字节                                     |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                       |    |     |
| 10    | <b>R9_R</b> : 区域9的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。   | 只读 | 0x1 |
| 9     | <b>R9_W</b> : 区域9的写入权限。                               | 只读 | 0x1 |
| 8     | <b>R9_X</b> : 区域9的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。   | 只读 | 0x1 |
| 7     | <b>R8_L</b> : 锁定区域8，且同时适用于M模式和U模式。                    | 只读 | 0x0 |
| 6:5   | 保留。                                                   | -  | -   |
| 4:3   | <b>R8_A</b> : 区域8的地址匹配类型。写入不支持的值（TOR）将使该区域被禁用。        | 只读 | 0x3 |
|       | 枚举值：                                                  |    |     |
|       | 0x0 → 关闭：禁用该区域                                        |    |     |
|       | 0x2 → NA4：自然对齐4字节                                     |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                       |    |     |
| 2     | <b>R8_R</b> : 区域8的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。   | 只读 | 0x1 |
| 1     | <b>R8_W</b> : 区域8的写入权限。                               | 只读 | 0x1 |
| 0     | <b>R8_X</b> : 区域8的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。   | 只读 | 0x1 |

## RVCSR：PMPCFG3寄存器

偏移：0x3a3

### 描述

区域12至区域15的物理内存保护配置

表388。PMPCFG3  
寄存器

| 位     | 描述                                                    | 类型 | 复位  |
|-------|-------------------------------------------------------|----|-----|
| 31    | <b>R15_L</b> : 锁定区域15，并同时应用于M模式与U模式。                  | 只读 | 0x0 |
| 30:29 | 保留。                                                   | -  | -   |
| 28:27 | <b>R15_A</b> : 区域15的地址匹配类型。写入不支持的值(TOR)将使该区域被禁用。      | 只读 | 0x0 |
|       | 枚举值：                                                  |    |     |
|       | 0x0 → 关闭：禁用该区域                                        |    |     |
|       | 0x2 → NA4：自然对齐4字节                                     |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                       |    |     |
| 26    | <b>R15_R</b> : 区域15的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x0 |
| 25    | <b>R15_W</b> : 区域15的写入权限。                             | 只读 | 0x0 |
| 24    | <b>R15_X</b> : 区域15的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x0 |
| 23    | <b>R14_L</b> : 锁定区域14，并同时应用于M模式与U模式。                  | 只读 | 0x0 |
| 22:21 | 保留。                                                   | -  | -   |
| 20:19 | <b>R14_A</b> : 区域14的地址匹配类型。写入不支持的值(TOR)将使该区域被禁用。      | 只读 | 0x0 |
|       | 枚举值：                                                  |    |     |
|       | 0x0 → 关闭：禁用该区域                                        |    |     |
|       | 0x2 → NA4：自然对齐4字节                                     |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                       |    |     |
| 18    | <b>R14_R</b> : 区域14的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x0 |
| 17    | <b>R14_W</b> : 区域14的写入权限。                             | 只读 | 0x0 |
| 16    | <b>R14_X</b> : 区域14的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x0 |
| 15    | <b>R13_L</b> : 锁定区域13，并同时应用于M模式及U模式。                  | 只读 | 0x0 |
| 14:13 | 保留。                                                   | -  | -   |
| 12:11 | <b>R13_A</b> : 区域13的地址匹配类型。写入不支持的值(TOR)将使该区域被禁用。      | 只读 | 0x0 |
|       | 枚举值：                                                  |    |     |
|       | 0x0 → 关闭：禁用该区域                                        |    |     |
|       | 0x2 → NA4：自然对齐4字节                                     |    |     |
|       | 0x3 → NAPOT：自然对齐的二次幂（8字节至4 GiB）                       |    |     |
| 10    | <b>R13_R</b> : 区域13的读取权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x0 |
| 9     | <b>R13_W</b> : 区域13的写入权限。                             | 只读 | 0x0 |
| 8     | <b>R13_X</b> : 区域13的执行权限。注意：由于RP2350-E6缺陷，R与X与标准位序相反。 | 只读 | 0x0 |

| 位   | 描述                                                      | 类型 | 复位  |
|-----|---------------------------------------------------------|----|-----|
| 7   | <b>R12_L</b> : 锁定区域12，并同时应用于M模式及U模式。                    | 只读 | 0x0 |
| 6:5 | 保留。                                                     | -  | -   |
| 4:3 | <b>R12_A</b> : 区域12的地址匹配类型。写入不支持的值(TOR)将使该区域被禁用。        | 只读 | 0x0 |
|     | 枚举值:                                                    |    |     |
|     | 0x0 → 关闭: 禁用该区域                                         |    |     |
|     | 0x2 → NA4: 自然对齐4字节                                      |    |     |
|     | 0x3 → NAPOT: 自然对齐的二次幂(8字节至4 GiB)                        |    |     |
| 2   | <b>R12_R</b> : 区域12的读取权限。注意: 由于RP2350-E6缺陷, R与X与标准位序相反。 | 只读 | 0x0 |
| 1   | <b>R12_W</b> : 区域12的写入权限。                               | 只读 | 0x0 |
| 0   | <b>R12_X</b> : 区域12的执行权限。注意: 由于RP2350-E6缺陷, R与X与标准位序相反。 | 只读 | 0x0 |

## RVCSR: PMPADDR0寄存器

偏移: 0x3b0

表389. PMPADDR0  
寄存器

| 位     | 描述                               | 类型 | 复位         |
|-------|----------------------------------|----|------------|
| 31:30 | 保留。                              | -  | -          |
| 29:0  | 第0区的物理内存保护地址。注意, 所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR: PMPADDR1寄存器

偏移: 0x3b1

表390. PMPADDR1  
寄存器

| 位     | 描述                               | 类型 | 复位         |
|-------|----------------------------------|----|------------|
| 31:30 | 保留。                              | -  | -          |
| 29:0  | 第1区的物理内存保护地址。注意, 所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR: PMPADDR2寄存器

偏移: 0x3b2

表391. PMPADDR2  
寄存器

| 位     | 描述                               | 类型 | 复位         |
|-------|----------------------------------|----|------------|
| 31:30 | 保留。                              | -  | -          |
| 29:0  | 第2区的物理内存保护地址。注意, 所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR: PMPADDR3寄存器

偏移: 0x3b3

表392. PMPADDR3  
寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:30 | 保留。 | -  | -  |

| 位    | 描述                              | 类型 | 复位         |
|------|---------------------------------|----|------------|
| 29:0 | 第3区的物理内存保护地址。注意，所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR：PMPADDR4寄存器

偏移: 0x3b4

表393. PMPADDR4 寄存器

| 位     | 描述                              | 类型 | 复位         |
|-------|---------------------------------|----|------------|
| 31:30 | 保留。                             | -  | -          |
| 29:0  | 区域4的物理内存保护地址。注意，所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR：PMPADDR5寄存器

偏移: 0x3b5

表394. PMPADDR5 寄存器

| 位     | 描述                              | 类型 | 复位         |
|-------|---------------------------------|----|------------|
| 31:30 | 保留。                             | -  | -          |
| 29:0  | 区域5的物理内存保护地址。注意，所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR：PMPADDR6寄存器

偏移: 0x3b6

表395. PMPADDR6 寄存器

| 位     | 描述                              | 类型 | 复位         |
|-------|---------------------------------|----|------------|
| 31:30 | 保留。                             | -  | -          |
| 29:0  | 区域6的物理内存保护地址。注意，所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR：PMPADDR7寄存器

偏移: 0x3b7

表396. PMPADDR7 寄存器

| 位     | 描述                              | 类型 | 复位         |
|-------|---------------------------------|----|------------|
| 31:30 | 保留。                             | -  | -          |
| 29:0  | 区域7的物理内存保护地址。注意，所有PMP地址单位均为4字节。 | 读写 | 0x00000000 |

## RVCSR：PMPADDR8寄存器

偏移: 0x3b8

表397. PMPADDR8 寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:30 | 保留。 | -  | -  |

| 位    | 描述                                                                                                                                                                                           | 类型 | 复位       |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|----------|
| 29:0 | <p>区域8的物理内存保护地址。注意，所有PMP地址单位均为4字节。</p> <p>硬连线至地址范围 <code>0x00000000</code> 至 <code>0x0fffffff</code>，其中包含启动只读存储器。默认情况下，此范围对用户模式开放访问。用户模式对该范围的访问可通过使用动态配置的PMP区域之一，或通过ACCESSCTRL中的权限寄存器禁用。</p> | 只读 | 0x01ffff |

## RVCSR：PMPADDR9寄存器

偏移量: 0x3b9

表398. PMPADDR9 寄存器

| 位     | 描述                                                                                                                                                                                        | 类型 | 复位       |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|----------|
| 31:30 | 保留。                                                                                                                                                                                       | -  | -        |
| 29:0  | <p>区域9的物理内存保护地址。注意，所有PMP地址单位均为4字节。</p> <p>硬连线至地址范围 <code>0x40000000</code> 至 <code>0x5fffffff</code>，其中包含系统外设。默认情况下，此范围对用户模式开放访问。用户模式对该范围的访问可通过使用动态配置的PMP区域之一，或通过ACCESSCTRL中的权限寄存器禁用。</p> | 只读 | 0x13ffff |

## RVCSR：PMPADDR10寄存器

偏移量: 0x3ba

表399。 PMPADDR10 寄存器

| 位     | 描述                                                                                                                                                                                                | 类型 | 复位       |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|----------|
| 31:30 | 保留。                                                                                                                                                                                               | -  | -        |
| 29:0  | <p>区域10的物理内存保护地址。注意，所有PMP地址单位均为4字节。</p> <p>硬连线至地址范围 <code>0xd0000000</code> 至 <code>0xdfffffff</code>，其中包含核心本地外设（SIO）。默认情况下，此范围对用户模式开放访问。用户模式对该范围的访问可通过使用动态配置的PMP区域之一，或通过ACCESSCTRL中的权限寄存器禁用。</p> | 只读 | 0x35ffff |

## RVCSR：PMPADDR11 寄存器

偏移: 0x3bb

表 400。  
PMPADDR11 寄存器

| 位     | 描述                                                          | 类型 | 复位         |
|-------|-------------------------------------------------------------|----|------------|
| 31:30 | 保留。                                                         | -  | -          |
| 29:0  | 第 11 区域的物理内存保护地址。注意，所有PMP地址单位均为4字节。<br><br>硬连线为全零值。该区域未被实现。 | 只读 | 0x00000000 |

## RVCSR: PMPADDR12 寄存器

偏移: 0x3bc

表 401。  
PMPADDR12 寄存器

| 位     | 描述                                                          | 类型 | 复位         |
|-------|-------------------------------------------------------------|----|------------|
| 31:30 | 保留。                                                         | -  | -          |
| 29:0  | 第 12 区域的物理内存保护地址。注意，所有PMP地址单位均为4字节。<br><br>硬连线为全零值。该区域未被实现。 | 只读 | 0x00000000 |

## RVCSR: PMPADDR13 寄存器

偏移: 0x3bd

表 402。  
PMPADDR13 寄存器

| 位     | 描述                                                          | 类型 | 复位         |
|-------|-------------------------------------------------------------|----|------------|
| 31:30 | 保留。                                                         | -  | -          |
| 29:0  | 第 13 区域的物理内存保护地址。注意，所有PMP地址单位均为4字节。<br><br>硬连线为全零值。该区域未被实现。 | 只读 | 0x00000000 |

## RVCSR: PMPADDR14 寄存器

偏移: 0x3be

表 403。  
PMPADDR14 寄存器

| 位     | 描述                                                       | 类型 | 复位         |
|-------|----------------------------------------------------------|----|------------|
| 31:30 | 保留。                                                      | -  | -          |
| 29:0  | 区域14的物理内存保护地址。注意，所有PMP地址单位均为4字节。<br><br>硬连线为全零值。该区域未被实现。 | 只读 | 0x00000000 |

## RVCSR: PMPADDR15 寄存器

偏移: 0x3bf

表 404。  
PMPADDR15 寄存器

| 位     | 描述                                                       | 类型 | 复位         |
|-------|----------------------------------------------------------|----|------------|
| 31:30 | 保留。                                                      | -  | -          |
| 29:0  | 区域15的物理内存保护地址。注意，所有PMP地址单位均为4字节。<br><br>硬连线为全零值。该区域未被实现。 | 只读 | 0x00000000 |

## RVCSR: TSELECT 寄存器

偏移: 0x7a0

表 405。TSELECT  
寄存器

| 位    | 描述                                                                                                      | 类型 | 复位  |
|------|---------------------------------------------------------------------------------------------------------|----|-----|
| 31:2 | 保留。                                                                                                     | -  | -   |
| 1:0  | 通过 <code>tdata1</code> / <code>tdata2</code> 配置的触发器选择<br><br>在 RP2350 上实现了四个指令地址触发器，因此该寄存器仅允许写入两个最低有效位。 | 读写 | 0x0 |

## RVCSR: TDATA1 寄存器

偏移: 0x7a1

### 描述

触发器配置数据1

Hazard 3 仅支持地址/数据匹配触发器 (type=2)，因此该寄存器描述包含此类型对应的 `mcontrol` 字段。

更准确地说，Hazard3 仅支持精确指令地址匹配触发器（硬件断点），因此本寄存器的许多字段均为固定连线。

表 406。TDATA1  
寄存器

| 位     | 描述                                                                                                                                                                                            | 类型 | 复位   |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|------|
| 31:28 | <b>TYPE:</b> 触发类型。硬连线为type=2，表示地址/数据匹配触发器。                                                                                                                                                    | 只读 | 0x2  |
| 27    | <b>D MODE:</b> 若为0，调试模式和M模式均可写入选定的 <code>tselect</code> 下的 <code>tdata</code> 寄存器。<br><br>若为1，仅调试模式可写入选定的 <code>tselect</code> 下的 <code>tdata</code> 寄存器。<br>其他模式的写入将被忽略。<br><br>该位仅可由调试模式写入。 | 读写 | 0x0  |
| 26:21 | <b>MASKMAX:</b> 值为0表示仅支持精确地址匹配。                                                                                                                                                               | 只读 | 0x00 |
| 20    | <b>HIT:</b> 触发命中标志。未实现，硬连线为0。                                                                                                                                                                 | 只读 | 0x0  |
| 19    | <b>SELECT:</b> 硬连线值为0表示仅支持地址匹配，不支持数据匹配。                                                                                                                                                       | 只读 | 0x0  |
| 18    | <b>TIMING:</b> 硬连线值为0表示触发在触发指令执行之前发生，而非之后。                                                                                                                                                    | 只读 | 0x0  |
| 17:16 | <b>SIZEL0:</b> 硬连线值为0表示不支持访问大小匹配。                                                                                                                                                             | 只读 | 0x0  |
| 15:12 | <b>ACTION:</b> 触发发生时选择执行的操作。                                                                                                                                                                  | 读写 | 0x0  |
|       | 枚举值：                                                                                                                                                                                          |    |      |

| 位    | 描述                                                    | 类型 | 复位  |
|------|-------------------------------------------------------|----|-----|
|      | 0x0 → EBREAK：引发断点异常，该异常可由M模式异常处理程序处理。                 |    |     |
|      | 0x1 → 调试：进入调试模式。仅当 tdata1.dmode为1时，方可选择此操作。           |    |     |
| 11   | <b>CHAIN</b> ：硬连线为0，表示不支持触发链。                         | 只读 | 0x0 |
| 10:7 | <b>MATCH</b> ：硬连线为0，表示匹配始终针对 <b>tdata2</b> 指定的完整地址有效。 | 只读 | 0x0 |
| 6    | <b>M</b> ：设置后，启用 M 模式下的此触发器。                          | 读写 | 0x0 |
| 5:4  | 保留。                                                   | -  | -   |
| 3    | <b>U</b> ：设置后，启用 U 模式下的此触发器。                          | 读写 | 0x0 |
| 2    | <b>EXECUTE</b> ：设置后，触发器将在执行指令的地址处触发。                  | 读写 | 0x0 |
| 1    | <b>STORE</b> ：硬连线为0，表示不支持存储地址/数据触发器。                  | 只读 | 0x0 |
| 0    | <b>LOAD</b> ：硬连线为0，表示不支持加载地址/数据触发器。                   | 只读 | 0x0 |

## RVCSR：TDATA2 寄存器

偏移：0x7a2

表 407. TDATA2 寄存器

| 位    | 描述                             | 类型 | 复位         |
|------|--------------------------------|----|------------|
| 31:0 | 触发配置数据2<br>包含用于指令地址触发（硬件断点）的地址 | 读写 | 0x00000000 |

## RVCSR：DCSR 寄存器

偏移：0x7b0

### 说明

调试控制与状态寄存器。在调试模式之外访问将触发非法指令异常。

表 408. DCSR 寄存器

| 位     | 描述                                                                                    | 类型 | 复位  |
|-------|---------------------------------------------------------------------------------------|----|-----|
| 31:28 | <b>XDEBUGVER</b> ：硬连线为4，符合RISC-V 0.13.2调试规范的外部调试支持。                                   | 只读 | 0x4 |
| 27:16 | 保留。                                                                                   | -  | -   |
| 15    | <b>EBREAKM</b> ：当该位为1时，M模式执行的 <b>ebreak</b> 指令将跳转至调试模式，而非产生陷阱。                        | 读写 | 0x0 |
| 14:13 | 保留。                                                                                   | -  | -   |
| 12    | <b>EBREAKU</b> ：当该位为1时，U模式执行的 <b>ebreak</b> 指令将跳转至调试模式，而非产生陷阱。                        | 读写 | 0x0 |
| 11    | <b>STEPIE</b> ：硬连线为0，硬件单步期间不接收中断。                                                     | 只读 | 0x0 |
| 10    | <b>STOPCOUNT</b> ：硬连线为1， <b>mcycle/mcycleh</b> 及 <b>minstret/minstreth</b> 在调试模式下不递增。 | 只读 | 0x1 |

| 位   | 描述                                                                      | 类型 | 复位  |
|-----|-------------------------------------------------------------------------|----|-----|
| 9   | <b>STOPTIME</b> : 硬连线为1，调试模式下核心本地计时器不递增。<br>外部计时器（例如hart共享计时器）可配置为忽略此项。 | 只读 | 0x1 |
| 8:6 | <b>CAUSE</b> : 由硬件在进入调试模式时设置。                                           | 只读 | 0x0 |
|     | 枚举值：                                                                    |    |     |
|     | 0x1 → EBREAK: 当相关的 dcsr.ebreakx 位被设置时，执行了 ebreak 指令。                    |    |     |
|     | 0x2 → TRIGGER: 触发模块引发断点异常。                                              |    |     |
|     | 0x3 → HALTREQ: 处理器因停止请求进入调试模式，或在释放核心复位时存在复位停止请求。                        |    |     |
|     | 0x4 → STEP: 启用单步执行后，处理器执行一条指令后进入调试模式。                                   |    |     |
| 5:3 | 保留。                                                                     | -  | -   |
| 2   | <b>STEP</b> : 为1时，处理器在M态或U态中执行每条指令后重新进入调试模式。<br>◦                       | 读写 | 0x0 |
| 1:0 | <b>PRV</b> : 读取进入调试模式时核心的特权模式，并设置从调试模式返回时核心执行的特权模式。                     | 读写 | 0x3 |

## RVCSR: DPC 寄存器

偏移量: 0x7b1

表409. DPC  
寄存器

| 位    | 描述                                                                                      | 类型 | 复位         |
|------|-----------------------------------------------------------------------------------------|----|------------|
| 31:1 | 调试程序计数器。进入调试模式时，dpc采样当前程序计数器，例如触发调试模式进入的 ebreak 指令地址。离开调试模式时，处理器跳转至dpc。主机在调试模式下可读写该寄存器。 | 读写 | 0x00000000 |
| 0    | 保留。                                                                                     | -  | -          |

## RVCSR: MCYCLE 寄存器

偏移: 0xb00

### 描述

机器模式周期计数器，低半部分

表 410. MCYCLE  
寄存器

| 位    | 描述                                         | 类型 | 复位         |
|------|--------------------------------------------|----|------------|
| 31:0 | 当mcountinhibit.cy为0时，每个时钟周期计数加一。默认禁用以节省功耗。 | 读写 | 0x00000000 |

## RVCSR: MINSTRET 寄存器

偏移: 0xb02

### 描述

机器模式已完成指令计数器，低半部分

表 411. MINSTRET  
寄存器

| 位    | 描述                                                        | 类型 | 复位         |
|------|-----------------------------------------------------------|----|------------|
| 31:0 | 当 <code>mcountinhibit.ir</code> 为 0 时，每条指令计数加一。默认禁用以节省功耗。 | 读写 | 0x00000000 |

## RVCSR: MHPMCOUNTER3、MHPMCOUNTER4、...、MHPMCOUNTER30， MHPMCOUNTER31 寄存器

偏移: 0xb03、0xb04、...、0xb1e、0xb1f

表 412.  
MHPMCOUNTER3、  
MHPMCOUNTER4、...、  
MHPMCOUNTER30  
、MHPMCOUNTER31  
寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 扩展性能计数器，硬连线为 0。 | 只读 | 0x00000000 |

## RVCSR: MCYCLEH 寄存器

偏移: 0xb80

### 描述

机器模式周期计数器，高半部分

表 413. MCYCLEH 寄存器

| 位    | 描述                                                                       | 类型 | 复位         |
|------|--------------------------------------------------------------------------|----|------------|
| 31:0 | 当 <code>mcountinhibit.cy</code> 为 0 时，每经过 $1 \ll 32$ 个周期计数加 1。默认禁用以节省功耗。 | 读写 | 0x00000000 |

## RVCSR: MINSTRETH 寄存器

偏移: 0xb82

### 描述

机器模式已完成指令计数器，低半部分

表 414。  
MINSTRETH 寄存器

| 位    | 描述                                                                        | 类型 | 复位         |
|------|---------------------------------------------------------------------------|----|------------|
| 31:0 | 当 <code>mcountinhibit.ir</code> 为 0 时，每经过 $1 \ll 32$ 条指令计数加 1。默认已禁用以节省电源。 | 读写 | 0x00000000 |

## RVCSR: MHPMCOUNTER3H、MHPMCOUNTER4H、...、MHPMCOUNTER30H， MHPMCOUNTER31H 寄存器

偏移: 0xb83, 0xb84, ..., 0xb9e, 0xb9f

表 415。  
MHPMCOUNTER3H  
、MHPMCOUNTER4H  
、...、MHPMCOUNT  
ER30H、MHPMCOU  
NTER31H 寄存器

| 位    | 描述              | 类型 | 复位         |
|------|-----------------|----|------------|
| 31:0 | 扩展性能计数器，硬连线为 0。 | 只读 | 0x00000000 |

## RVCSR: PMPCFGM0 寄存器

偏移量: 0xbd0

表 416。  
PMPCFGM0 寄存器

| 位     | 描述  | 类型 | 复位 |
|-------|-----|----|----|
| 31:16 | 保留。 | -  | -  |

| 位    | 描述                                                                                                                                                                                                                                                                                                                                                       | 类型 | 复位     |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|--------|
| 15:0 | <p>PMP M 模式配置。每个 PMP 区域对应一位。设置该位后，相应区域将应用于 M 模式（类似 <code>pmpcfg.L</code> 位），但不会锁定该区域。</p> <p>PMP 适用于非安全相关用途，例如堆栈保护和外设仿真。该扩展允许 M 模式自由使用当前所有未锁定区域，用于自身目的，无需锁定带来的限制。</p> <p>请注意，这并不授予 M 模式新的权限，因为在基础标准中，已可通过锁定将未锁定区域应用于 M 模式。通常，应按区域编号升序锁定 PMP 区域，防止后续未锁定区域覆盖已锁定区域。</p> <p>另请注意，此规则不同于 ePMP 扩展中的规则锁定绕过位，后者不允许已锁定与未锁定的 M 模式区域共存。</p> <p>这是Hazard3定制的CSR。</p> | 读写 | 0x0000 |

## RVCSR：MEIEA 寄存器

偏移量: 0xbe0

### 说明

外部中断使能数组。

该数组包含每个外部中断请求的可读写位：值为 1 表示该中断目前处于使能状态。  
复位时，所有外部中断均被禁用。

使能时，外部中断可触发标准RISC-V机器外部中断挂起标志  
(`mip.meip`)，从而导致处理器进入外部中断向量。详见 `meipa`。

最多支持512个外部中断。本寄存器的高半部分包含对完整512位向量的16位窗口。窗口索引由写入数据的最低5位决定。

表417. MEIEA  
寄存器

| 位     | 描述                                                            | 类型 | 复位     |
|-------|---------------------------------------------------------------|----|--------|
| 31:16 | <b>WINDOW:</b> 外部中断使能数组的16位读写窗口                               | 读写 | 0x0000 |
| 15:5  | 保留。                                                           | -  | -      |
| 4:0   | <b>INDEX:</b> 仅写自清字段（不保存值），用于控制 <code>window</code> 中显示的数组窗口。 | WO | 0x00   |

## RVCSR：MEIPA 寄存器

偏移量: 0xbe1

### 描述

外部中断挂起数组

包含针对每个外部中断请求的只读位。与 `meiea` 类似，该寄存器是最多 512 个外部中断标志数组的窗口。状态显示在从 `meip` 读取值的高 16 位，且同一 CSR 指令写入的值的低 5 位（若未写入则为 0）用于选择完整中断挂起数组的 16 位窗口。

位 1 表示该中断当前处于断言状态。中断请求 (IRQ) 假定为电平敏感，相应的 `meipa` 位通过服务请求方清除，以取消其中断请求的断言。

当任何优先级足够的中断在meipa中被设置且在meiea中被使能时，标准的RISC-V外部中断挂起位mip.meip将被断言。

换言之，`meipa`经`meiea`过滤后生成标准的`mip.meip`标志。换言之，`meipa`经由`meiea`过滤以生成标准`mip.meip`标志。

表 418. MEIPA  
寄存器

| 位     | 描述                                     | 类型 | 复位   |
|-------|----------------------------------------|----|------|
| 31:16 | 窗口：指向外部中断挂起数组的16位只读窗口                  | 只读 | -    |
| 15:5  | 保留。                                    | -  | -    |
| 4:0   | 索引：仅写且自动清除字段（不存储数值），用于控制数组的哪个窗口显示在窗口中。 | WO | 0x00 |

## RVCSR：MEIFA 寄存器

偏移：0xbe2

### 描述

外部中断强制数组

包含每个中断请求的读写位。向中断强制数组中的某个位写入1，将导致对应位在`meipa`中变为挂起状态。软件可利用此功能手动触发特定中断。

在中断处理程序中使用`meifa`没有任何限制。更为实用的情况是在高优先级中断中调度低优先级处理程序，以便其在核心返回前台代码之前执行。实现者可考虑保留部分外部IRQ，其外部输入固定接地（连接至0）以供此用途。

相关位可由软件清除，并且在读取`meinext`时会由硬件自动清除，读取时返回对应的IRQ编号于`meinext.irq`，同时使用`minext.noirq`清除（无论是否写入`meinext.update`）。

`meifa`实现了与`meiea`和`meipa`相同的数组窗口索引方案。

表 419. MEIFA  
寄存器

| 位     | 描述                                     | 类型 | 复位     |
|-------|----------------------------------------|----|--------|
| 31:16 | <b>WINDOW</b> ：对外部中断强制数组的16位读/写窗口      | 读写 | 0x0000 |
| 15:5  | 保留。                                    | -  | -      |
| 4:0   | 索引：仅写且自动清除字段（不存储数值），用于控制数组的哪个窗口显示在窗口中。 | WO | 0x00   |

## RVCSR：MEIPRA 寄存器

偏移：0xbe3

### 描述

外部中断优先级数组

每个中断均关联一个（最多）4位优先级值，对该寄存器的每次访问均读取和/或写入包含四个此类优先级值的16位窗口。当可用优先级级别少于16级时，优先级字段的最低有效位固定为0。

当中断优先级低于当前抢占优先级`meicontext.preempt`时，针对`mip.meip`，该中断将视为未挂起状态。`meipa`中的挂起位仍会置位，但机器外部中断挂起位`mip.meip`不会置位，因此处理器将忽略该中断。详见`meicontext`。

表 420. MEIPRA  
寄存器

| 位     | 描述                                           | 类型 | 复位     |
|-------|----------------------------------------------|----|--------|
| 31:16 | <b>WINDOW</b> ：外部中断优先级数组的16位读写窗口，包含四个4位优先级值。 | 读写 | 0x0000 |
| 15:5  | 保留。                                          | -  | -      |

| 位   | 描述                                     | 类型 | 复位   |
|-----|----------------------------------------|----|------|
| 4:0 | 索引：仅写且自动清除字段（不存储数值），用于控制数组的那个窗口显示在窗口中。 | WO | 0x00 |

## RVCSR: MEINEXT寄存器

偏移: 0xbe4

### 描述

获取下一个外部中断

包含在 `meipa` 中断请求且在 `meiea` 中使能的最高优先级外部中断的索引，左移2位，以用于索引32位函数指针数组。如果不存在此类中断，则最高有效位（MSB）被置位。

当多个相同优先级的中断既处于挂起状态又被使能时，编号最低者优先。优先级低于 `meicontext.ppreempt` ——前一个抢占优先级——的中断，视同未挂起。

此机制确保抢占中断帧不会处理中断帧中可能正在执行的中断。

表421. MEINEXT 寄存器

| 位     | 描述                                                                                                                                                      | 类型 | 复位    |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------|----|-------|
| 31    | <b>NOIRQ</b> : 当不存在已启用、挂起且优先级大于或等于 <code>meicontext.ppreempt</code> 的外部中断时设置。可通过 <code>bltz</code> 或 <code>bgez</code> 指令高效测试。                          | 只读 | 0x0   |
| 30:11 | 保留。                                                                                                                                                     | -  | -     |
| 10:2  | <b>IRQ</b> : 最高优先级活动外部中断的索引。当不存在满足优先级条件且已挂起且已启用的外部中断时，值为零。                                                                                              | 只读 | 0x000 |
| 1     | 保留。                                                                                                                                                     | -  | -     |
| 0     | <b>UPDATE</b> : 写入1（自清除）将根据 <code>noirq/irq</code> 中指示的中断编号和抢占优先级，硬件更新 <code>meicontext</code> 。此操作应作为单个原子操作完成，即 <code>csrrsi a0, meinext, 0x1</code> 。 | SC | 0x0   |

## RVCSR: MEICONTEXT寄存器

偏移: 0xbe5

### 描述

外部中断上下文寄存器

配置中断抢占的优先级等级，帮助软件跟踪当前所处的中断状态。当通用中断服务程序处理来自同一外设多个实例的中断请求时，此功能尤为重要。

在 `preempt`、`ppreempt` 和 `pppreempt` 字段中维护了三级抢占优先级栈。当硬件进入外部中断向量时，优先级栈会被保存；若设置了 `meicontext.mreteirq`，则通过 `mret` 指令恢复该栈。

优先级栈的顶部条目 `preempt` 被硬件用以确保仅更高优先级的中断能够抢占当前中断。下一个条目 `ppreempt` 用于避免在已被抢占的帧中服务的中断被重复处理。第三个条目 `pppreempt` 无硬件效应，但确保 `preempt` 和 `ppreempt` 可以在任意抢占层级间被正确保存与恢复。

表422。  
MEICONTEXT寄存器

| 位     | 描述                                                                                                                                                                                                                                                                                                                                                                     | 类型 | 复位    |
|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-------|
| 31:28 | <b>PPPREEMPT:</b> 先前的 <code>ppreempt</code> 。优先级保存时设为 <code>ppreempt</code> , 优先级恢复时设为零。无硬件效应, 但确保当 <code>meicontext</code> 被正确保存和恢复时, <code>preempt</code> 及 <code>ppreempt</code> 在任意多抢占帧中能正确维护栈状态。                                                                                                                                                                  | 读写 | 0x0   |
| 27:24 | <b>PPREEMPT:</b> 先前的 <code>preempt</code> 。在优先级保存时设置为 <code>preempt</code> , 优先级恢复时恢复为 <code>ppreempt</code> 。<br><br>低于 <code>ppreempt</code> 优先级的 IRQ 不会在 <code>meinext</code> 中显示, 以防止在抢占帧中重新抢占被抢占者。                                                                                                                                                                | 读写 | 0x0   |
| 23:21 | 保留。                                                                                                                                                                                                                                                                                                                                                                    | -  | -     |
| 20:16 | <b>PREEMPT:</b> 抢占当前中断的最低中断优先级。<br>低于 <code>preempt</code> 优先级的中断不会导致核心跳转至中断处理程序。写入 <code>meinext.update</code> 或硬件进入外部中断向量时由硬件更新。<br><br>如果此字段更新时 <code>meinext</code> 中存在中断, 则 <code>preempt</code> 被设置为该中断优先级高一级。否则, <code>ppreempt</code> 被设置为高于最大中断优先级一级, 禁止抢占。                                                                                                   | 读写 | 0x00  |
| 15    | <b>NOIRQ:</b> 非中断状态 (读/写)。复位时设置为 1。写入 <code>meinext.update</code> 时将设置为 <code>meinext.noirq</code> 。无硬件影响。                                                                                                                                                                                                                                                             | 读写 | 0x1   |
| 14:13 | 保留。                                                                                                                                                                                                                                                                                                                                                                    | -  | -     |
| 12:4  | <b>IRQ:</b> 当前 IRQ 编号 (可读写)。在写入 <code>meinext.update</code> 时设置为 <code>meinext.irq</code> 。无硬件影响。                                                                                                                                                                                                                                                                      | 读写 | 0x000 |
| 3     | <b>MTIESAVE:</b> 如果 <code>clearts</code> 由同一 CSR 访问指令设置, 则读取为 <code>mie.mtie</code> 的当前值; 否则读取为 0。写入值将与 <code>mie.mtie</code> 执行逻辑或操作。                                                                                                                                                                                                                                 | 只读 | 0x0   |
| 2     | <b>MSIESAVE:</b> 如果 <code>clearts</code> 由同一 CSR 访问指令设置, 则读取为 <code>mie.msie</code> 的当前值; 否则读取为 0。写入值将与 <code>mie.msie</code> 执行逻辑或操作。                                                                                                                                                                                                                                 | 只读 | 0x0   |
| 1     | <b>CLEARTS:</b> 写 1 即自清字段。写入 1 将清除 <code>mie.mtie</code> 和 <code>mie.msie</code> , 并在该寄存器的 <code>mtiesave</code> 和 <code>msiesave</code> 字段中保存其先前值。此操作确保通过 <code>mstatus.mie</code> 重新启用 IRQ 时安全无虞, 避免被标准定时器及软中断处理程序抢占, 而这些程序可能不识别 Hazard3 中断硬件。<br><br>由于 <code>clearts</code> 导致的清除优先于由于 <code>mtiesave</code> / <code>msiesave</code> 设置的清除, 尽管软件在同一周期内写入两者的情况异常罕见。 | SC | 0x0   |
| 0     | <b>MRETEIRQ:</b> 如果为 1, 则启用在 <code>mret</code> 指令上恢复抢占优先级堆栈。此位在进入外部中断向量时被置位, 在执行 <code>mret</code> 时清除, 并且在触发任何非外部中断陷阱时亦被清除。<br><br>前提是 <code>meicontext</code> 在进入外部中断向量时被保存 (在启用抢占之前)、在退出前被恢复, 且通过使用 <code>clearts</code> 等方式防止标准软件/定时器 IRQ 的抢占, 本标志允许硬件即使在外部中断处理程序可能引发异常的情况下, 也能安全管理抢占优先级堆栈。                                                                        | 读写 | 0x0   |

## RVCSR: MSLEEP 寄存器

偏移量: 0xbf0

**描述**

M 模式休眠控制寄存器

表423. MSLEEP 寄存器

| 位    | 描述                                                                                                                                                                                  | 类型 | 复位  |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:3 | 保留。                                                                                                                                                                                 | -  | -   |
| 2    | <b>SLEEPONBLOCK</b> : 在执行 <code>h3.block</code> 指令以及标准 <code>wfi</code> 时，进入由 <code>msleep.deepsleep/msleep.powerdown</code> 配置的深度睡眠状态。如果该位清零， <code>h3.block</code> 始终作为简单流水线阻塞实现。 | 读写 | 0x0 |
| 1    | <b>POWERDOWN</b> : 进入休眠时释放外部电源请求。此功能由平台定义——可能无任何操作，可能执行简单操作如时钟门控，亦可能与复杂系统级电源控制器相关联。<br><br>唤醒时，处理器重新断言外部上电请求，并在请求获得确认前不获取任何指令。这可能显著增加唤醒延迟。                                          | 读写 | 0x0 |
| 0    | <b>DEEPSLEEP</b> : 进入休眠状态时取消处理器时钟使能。如果实例化了时钟门控，允许在睡眠时对大部分处理器（除电源状态机及中断与停止输入寄存器外）实施时钟门控，从而降低睡眠电流。<br><br>这会增加一个时钟周期的唤醒延迟。                                                            | 读写 | 0x0 |

**RVCSR： DMDATA0 寄存器**

偏移: 0xbff

表 424. DMDATA0 寄存器

| 位    | 描述                                                                                                   | 类型 | 复位         |
|------|------------------------------------------------------------------------------------------------------|----|------------|
| 31:0 | 调试模块的DATA0寄存器映射至Hazard3的CSR空间，调试模块通过执行CSR访问指令与处理器核心交换数据（此机制用于实现Abstract Access Register指令）。仅限调试模式访问。 | 读写 | 0x00000000 |

**RVCSR： CYCLE 寄存器**

偏移: 0xc00

表425. CYCLE 寄存器

| 位    | 描述                                                   | 类型 | 复位         |
|------|------------------------------------------------------|----|------------|
| 31:0 | mcycle的只读用户模式别名，访问条件是 <code>mcounteren.cy</code> 被设置 | 只读 | 0x00000000 |

**RVCSR： INSTRET 寄存器**

偏移: 0xc02

表426. INSTRET 寄存器

| 位    | 描述                                                     | 类型 | 复位         |
|------|--------------------------------------------------------|----|------------|
| 31:0 | minstret的只读用户模式别名，访问条件是 <code>mcounteren.ir</code> 被设置 | 只读 | 0x00000000 |

**RVCSR： CYCLEH 寄存器**

偏移: 0xc80

表427. CYCLEH  
寄存器

| 位    | 描述                                     | 类型 | 复位         |
|------|----------------------------------------|----|------------|
| 31:0 | mcycleh的只读用户模式别名，访问条件是mcounteren.cy被设置 | 只读 | 0x00000000 |

## RVCSR：INSTRETH寄存器

偏移: 0xc82

表428. INSTRETH  
寄存器

| 位    | 描述                                       | 类型 | 复位         |
|------|------------------------------------------|----|------------|
| 31:0 | minstreth 的只读用户模式别名，在设置mcounteren.ir后可访问 | 只读 | 0x00000000 |

## RVCSR：MVENDORID 寄存器

偏移: 0xf11

### 描述

厂商 ID

表 429。  
MVENDORID 寄存器

| 位    | 描述                                                                    | 类型 | 复位         |
|------|-----------------------------------------------------------------------|----|------------|
| 31:7 | 银行: 值为 9 表示 9 个延续码，对应 JEP106 第10号银行。                                  | 只读 | 0x00000009 |
| 6:0  | 偏移: 第10号银行中 ID 为 0x13 的值，是 Raspberry Pi Ltd (RP2350 供应商) 的 JEP106 ID。 | 只读 | 0x13       |

## RVCSR：MARCHID 寄存器

偏移: 0xf12

表 430。MARCHID  
寄存器

| 位    | 描述              | 类型 | 复位          |
|------|-----------------|----|-------------|
| 31:0 | 架构标识符 (Hazard3) | 只读 | 0x00000001b |

## RVCSR：MIMPID 寄存器

偏移: 0xf13

表 431。MIMPID  
寄存器

| 位    | 描述                                                   | 类型 | 复位         |
|------|------------------------------------------------------|----|------------|
| 31:0 | 实现标识。在RP2350上，该值为0x86fc4e3f，表示Hazard3版本v1.0-rc1<br>。 | 只读 | 0x86fc4e3f |

## RVCSR：MHARTID 寄存器

偏移: 0xf14

### 描述

硬件线程 ID

表432. MHARTID  
寄存器

| 位    | 描述                                    | 类型 | 复位 |
|------|---------------------------------------|----|----|
| 31:0 | 在RP2350上，核心0的hart ID为0，核心1的hart ID为1。 | 只读 | -  |

## RVCSR：MCONFIGPTR寄存器

偏移: 0xf15

表433。  
MCONFIGPTR寄存器

| 位    | 描述                | 类型 | 复位         |
|------|-------------------|----|------------|
| 31:0 | 指向配置数据结构的指针（固定为0） | 只读 | 0x00000000 |

## 3.9. Arm/RISC-V 架构切换

RP2350 支持 Arm 和 RISC-V 两种处理器架构。基于 SDK 的程序通常不包含汇编代码，通过提供适当的构建标志，即可在任一架构上无修改运行。

RP2350 配备两个处理器插槽，在本文档中分别称为核心0和核心1。每个插槽可由 Cortex-M33 处理器（实现 Armv8-M 主架构及扩展）或 Hazard3 处理器（实现 RV32IMAC 架构及扩展）占用。

当处理器复位解除时，硬件会采样 OTP 控制寄存器块中的 ARCHSEL 寄存器，以确定连接至该插槽的处理器类型。未使用的处理器将无限期保持复位状态，其时钟输入被门控。ARCHSEL 寄存器的默认值及允许值由关键 OTP 标志决定：

- 如果 CRIT0\_ARM\_DISABLE 被设置，则仅允许使用 RISC-V。
- 否则，如果 CRIT0\_RISCV\_DISABLE 被设置，则仅允许使用 Arm。
- 否则，如果 CRIT1\_SECURE\_BOOT\_ENABLE 被设置，则仅允许使用 Arm。
- 否则，如果 CRIT1\_BOOT\_ARCH 被设置，则两种架构均被允许，且默认使用 RISC-V。
- 如果上述标志均未设置，则两种架构均被允许，且默认使用 Arm。

默认情况下，未设置任何 CRIT1 标志，因此在同时支持两种架构的设备上，默认架构为 Arm。要将默认架构更改为 RISC-V，请将 CRIT1\_BOOT\_ARCH 标志设置为 1。

启用安全启动后，RISC-V 内核将被禁用，因为 RP2350 启动只读存储器（bootrom）未对 RISC-V 实施安全启动。这有效防止恶意行为者通过切换架构绕过安全启动机制。

### ① 注意

自 RP2350 A3 版本起，CRIT0\_ARM\_DISABLE 标志不再生效，消除了在受保护的 RP2350 上调试时可能存在的解锁路径。此外，CRIT0\_RISCV\_DISABLE=1 与 CRIT1\_BOOT\_ARCH=1 的组合被解码为无效状态，阻止了启动。

RP2350 仅在处理器复位时采样 ARCHSEL 寄存器。该寄存器在其他时间点的值将被忽略，因此软件可在看门狗复位前对其进行编程，以实现软件发起的架构切换。

读取 ARCHSEL\_STATUS 寄存器以检查各处理器最近采样的 ARCHSEL 值。

### 3.9.1. 自动切换

RP2350 的二进制文件包含由只读存储器引导程序识别的二进制标记。该标记：

- 包含二进制入口点及目标架构（Arm、RISC-V 或两者）的附加信息；
- 帮助检测闪存设备是否已连接；
- 帮助验证访问闪存设备时是否采用了正确的 SPI 参数；

当核心 0 以 Arm 架构模式启动时，若检测到可引导的 RISC-V 二进制文件，引导只读存储器会自动重置两个核心，并切换为 RISC-V 架构模式。复位后，bootrom 检测到二进制文件与处理器架构匹配，因而正常启动该二进制文件。

同样，当以 RISC-V 架构模式在核心 0 启动时，bootrom 在检测到可启动的 Arm 二进制文件后，会自动复位两个核心，并将其切换至 Arm 架构模式。

因此，运行于Arm和RISC-V架构上的USB引导加载程序能够接受任一架构的UF2镜像下载，并自动使用正确的处理器启动该镜像。

### 3.9.2. 混合架构组合

ARCHSEL寄存器为每个处理器插槽分配一位，因此可以请求Arm与RISC-V处理器的混合组合：例如Arm核心0与RISC-V核心1，或RISC-V核心0与Arm核心1。

实际应用受限，因为这需要两个独立的程序镜像。两个核心能够正常协作，包括通过全局监视器实现的共享排他访问：例如，Arm处理器执行 `ldrex` 和 `strex` 指令，RISC-V处理器执行 `amoadd.w` 指令时，能安全并发访问共享变量。

硬件支持Arm和RISC-V处理器的混合调试，尽管这在主机软件端可能存在一定挑战。未使用处理器的调试资源在顶层Core Sight只读存储器(ROM)表中动态标记为非PRESENT状态。

# 第4章 内存

RP2350内置只读存储器(ROM)、一次性可编程存储器(OTP)和静态随机存取存储器(SRAM)。RP2350通过QSPI接口提供对外部闪存的访问。

## 4.1. 只读存储器

一个32 kB只读存储器(ROM)位于地址 `0x00000000`。ROM内容在硅片制造时永久固定。第5章详细描述了ROM内容，简而言之，包括：

- 核心0启动代码（第5.2节）
- 核心1启动代码（第5.3节）
- 运行时API（第5.4节）。
- USB引导加载程序
  - 用于拖拽UF2闪存和SRAM二进制文件的大容量存储接口（第5.5节）
  - PICOBLOCK接口，支持 `picotool` 及OTP编程等高级操作（第5.6节）
  - 支持对白标化所有通过USB公开的信息/标识符（第5.7节）
- UART引导加载程序：用于从主机微控制器加载SRAM二进制文件的最小化shell（第5.8节）。

只读存储器提供单周期访问，并配备专用的AHB5仲裁器，因此可与其他存储设备同时访问。对只读存储器的写入无任何效果，且写入时不会产生总线错误。

只读存储器受第10.2.2节中列举的IDAU区域保护。该保护机制有助于将引导只读存储器划分为安全与非安全代码：尤其是USB/UART引导加载程序以非安全客户端应用程序身份在Arm上运行，以减少安全引导实现的攻击面。

部分只读存储器功能未在RISC-V上实现，尤以安全引导功能最为显著。

## 4.2. 静态随机存取存储器

片上SRAM总容量为520 kB（ $520 \times 1024$ 字节）。出于性能考虑，物理上将其划分为十个bank，但逻辑上仍表现为单一连续的520 kB内存。RP2350对每个bank中存储的数据不加限制：可以使用任何bank存储处理器代码、数据缓冲区或二者的混合。共有八个 $16,384 \times 32$ 位的bank（每个64 kB）和两个 $1024 \times 32$ 位的bank（每个4 kB）。

### 注意

Banking是对SRAM的物理分区，通过支持多路并行访问以提升性能。逻辑上，内存呈现为单一连续的520 kB空间。

每个SRAM bank均通过专用的AHB5仲裁器进行访问。这意味着不同的总线管理器可并行访问不同的SRAM bank，从而每个系统时钟周期最多可执行六次32位SRAM访问（每个管理器一次）。

SRAM在系统地址中的映射起始于 `0x20000000`。首个256 kB地址区域，直至包括 `0x2003ffff`，按照字(word)条纹方式分布于前四个64 kB bank。接下来的256 kB地址区域，直到 `0x2007ffff` 按字分条跨越剩余的四个64 kB存储区。这两个分条区域之间的分界水印，位于 `0x20040000`，标志着SRAM0和SRAM1电源域的边界。

系统地址空间中的连续字按照表434所示路由到不同的RAM存储区。该方案称为顺序交错，能够提升典型内存访问模式下的总线并行度。

表434。SRAM  
存储区0/1/2/3交错  
映射。

| 系统地址       | SRAM存储区 | SRAM字地址 |
|------------|---------|---------|
| 0x20000000 | 存储区0    | 0       |
| 0x20000004 | 存储区1    | 0       |
| 0x20000008 | 存储区2    | 0       |
| 0x2000000c | 存储区3    | 0       |
| 0x20000010 | 存储区0    | 1       |
| 0x20000014 | 存储区1    | 1       |
| 0x20000018 | 存储区2    | 1       |
| 0x2000001c | 存储区3    | 1       |
| 0x20000020 | 存储区0    | 2       |
| 0x20000024 | 存储区1    | 2       |
| 0x20000028 | 存储区2    | 2       |
| 0x2000002c | 存储区3    | 2       |
| 等等         |         |         |

最高的两个4 KB区域（起始地址为 0x20080000 和 0x20081000）直接映射到较小的4 KB内存块。

软件可选择将这些区域用于每核用途（例如堆栈和频繁执行的代码），以确保处理器在此类访问中绝不阻塞。如同RP2350上的所有SRAM，这些内存块可被所有管理器单周期访问（前提是不同时有其他管理器在同一周期访问该内存块），因此合理地将内存作为单一520 KB设备处理。

### ① 注意

RP2040拥有非条带式SRAM镜像。RP2350不再拥有非条带式镜像，以避免将同一SRAM地址映射为安全和非安全两种状态。您仍可通过在两个256 KB的四路条带SRAM块间分配数据，实现明确的带宽划分。

## 4.2.1. 其他片上存储器

除520 kB主存储器外，还有两个专用RAM块，在某些情况下可用：

- 缓存行可在XIP地址空间内单独固定，用作SRAM，最高可达16 kB缓存总容量（参见第4.4.1.3节）。未固定的缓存行仍可用于透明缓存XIP访问。
- 若不使用USB，则USB数据DPRAM可用作起始地址为 0x50100000 的4 kB内存。

还有1 kB专用启动RAM，仅限安全访问，其内容和布局由只读存储器定义——详见第5章。

### 注意

外设地址空间内的内存（地址以 `0x4`、`0x5` 或 `0xd` 开头）不支持代码执行。这包括USB RAM和启动RAM。为了简化通过ACCESSCTRL将外设分配至安全域的操作，这些地址范围被设为IDAU免除，因而必须设置为不可执行，以避免出现非安全可写且安全可执行内存的情况。

## 4.3. 启动RAM

Boot RAM 是一个 1 kB (256 × 32 位) SRAM，专用于 bootrom。其速度低于主 SRAM，因为通过 APB 总线访问，读取需三周期，写入需四周期。

Boot RAM 在启动过程中用途广泛，包括初始预启动堆栈。在 bootrom 进入用户应用程序后，boot RAM 保存面向用户的只读存储器 API 的状态信息，如用于闪存编程保护的常驻分区表，以及闪存 XIP 设置函数的副本（前称为 `boot2`），以便在串行编程操作后快速重新初始化闪存 XIP 模式。

Boot RAM 硬连线仅允许安全访问（Arm 架构）或机器模式访问（RISC-V 架构）。无论 MPU 配置如何，均无法从 boot RAM 执行代码，因为其位于 APB 外设总线段，且未连接至处理器指令取指端口。

由于启动RAM位于XIP RAM电源域内，因此只要切换的核心域通电，启动RAM始终保持通电状态。这简化了bootrom中的SRAM电源管理，因为在找到存储调用栈的位置之前，无需为任何RAM供电。

启动RAM支持RP2350其他外设所采用的标准原子设定/清除/XOR访问（第2.1.3节）。

虽然可以将启动RAM用于用户自定义用途，但不建议这样做，因为这可能导致只读存储器API行为异常。调用只读存储器可能会修改存储于启动RAM中的数据。

### 4.3.1. 寄存器列表

少数寄存器与启动RAM共享相同的总线端点：

#### 一次写入位

这些标志一旦设置，仅能通过系统复位清除。它们用于实现某些bootrom安全特性。

#### 启动锁

其功能与SIO自旋锁相同（第3.1.4节），但通常专用于bootrom（第5.4.4节）。

这些寄存器起始于引导RAM基址 `0x400e0000`（在SDK中定义为`BOOTRAM_BASE`）以上的偏移量 `0x800`处。

表435。BOOTRAM 寄存器列表

| 偏移量   | 名称                         | 说明                                                       |
|-------|----------------------------|----------------------------------------------------------|
| 0x800 | <code>WRITE_ONCE0</code>   | 该寄存器写入时总是对其当前内容执行位或操作。一旦某个位被置位，只能通过复位来清除。                |
| 0x804 | <code>WRITE_ONCE1</code>   | 该寄存器写入时总是对其当前内容执行位或操作。一旦某个位被置位，只能通过复位来清除。                |
| 0x808 | <code>BOOTLOCK_STAT</code> | Bootlock状态寄存器。1表示未声明，0表示已声明。这些锁的功能与SIO旋转锁相同，但专用于bootrom。 |

| 偏移量   | 名称        | 说明                                            |
|-------|-----------|-----------------------------------------------|
| 0x80c | BOOTLOCK0 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x810 | BOOTLOCK1 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x814 | BOOTLOCK2 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x818 | BOOTLOCK3 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x81c | BOOTLOCK4 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x820 | BOOTLOCK5 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x824 | BOOTLOCK6 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |
| 0x828 | BOOTLOCK7 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 |

## BOOTRAM: WRITE\_ONCE0、WRITE\_ONCE1 寄存器

偏移量: 0x800, 0x804

表 436.  
WRITE\_ONCE0,  
WRITE\_ONCE1  
寄存器

| 位    | 描述                                        | 类型 | 复位         |
|------|-------------------------------------------|----|------------|
| 31:0 | 该寄存器写入时总是对其当前内容执行位或操作。一旦某个位被置位，只能通过复位来清除。 | 读写 | 0x00000000 |

## BOOTRAM: BOOTLOCK\_STAT 寄存器

偏移量: 0x808

表 437.  
BOOTLOCK\_STAT  
寄存器

| 位    | 描述                                                       | 类型 | 复位   |
|------|----------------------------------------------------------|----|------|
| 31:8 | 保留。                                                      | -  | -    |
| 7:0  | Bootlock状态寄存器。1表示未声明，0表示已声明。这些锁的功能与SIO旋转锁相同，但专用于bootrom。 | 读写 | 0xff |

## BOOTRAM: BOOTLOCK0、BOOTLOCK1、...、BOOTLOCK6、BOOTLOCK7 寄存器

偏移量: 0x80c、0x810、...、0x824、0x828

表 438。  
BOOTLOCK0,  
BOOTLOCK1, ...  
BOOTLOCK6,  
BOOTLOCK7  
寄存器

| 位    | 描述                                            | 类型 | 复位         |
|------|-----------------------------------------------|----|------------|
| 31:0 | 读取以声明和检测，写入以取消声明。声明成功时返回值为 $1 \ll n$ ，失败时返回0。 | 读写 | 0x00000000 |

## 4.4. 外部闪存和PSRAM (XIP)

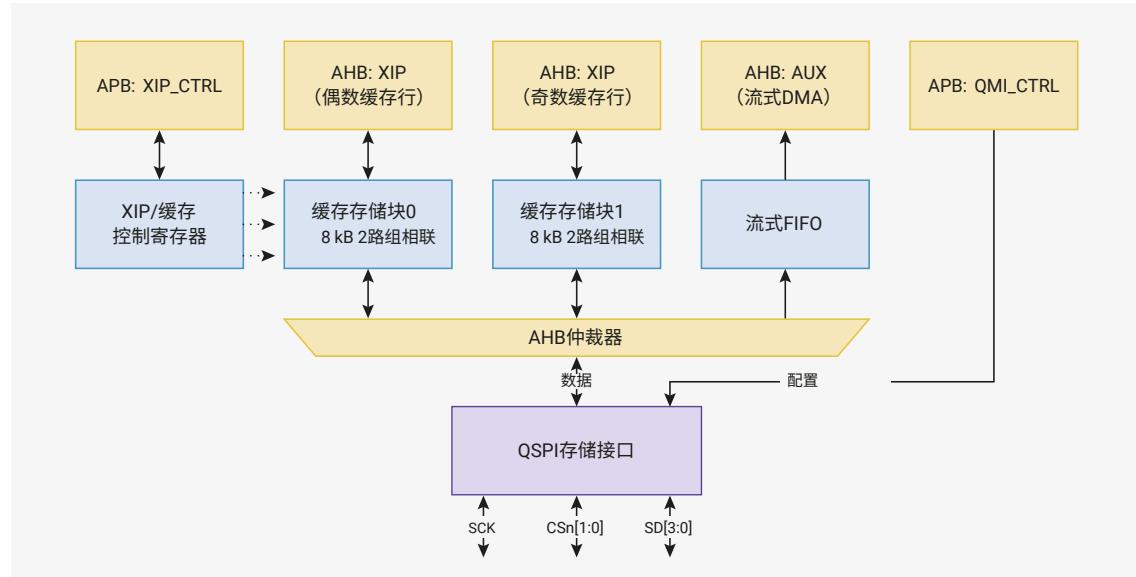
RP2350可通过其**execute-in-place** (XIP) 子系统访问外部闪存和PSRAM。术语**execute-in-place**指外部存储器直接映射到芯片内部地址空间。这使您能够执行

来自外部存储器的代码，而无需显式复制至片上SRAM。例如，处理器从AHB地址 `0x10001234` 访问指令时，会导致QSPI存储器接口从外部闪存设备地址 `0x001234` 读取数据。

片上16 kB缓存保存最近读取和写入的数据。这减少了XIP总线访问必须访问外部存储器的次数，提高了XIP接口的平均吞吐量和延迟性能。缓存在物理上结构为两个8 kB的存储块，奇偶缓存行以8字节粒度交错分布于两块存储块中。这使得处理器能够在同一时钟周期内访问多个缓存行。逻辑上，XIP缓存表现为单一16 kB缓存。

图16。闪存

执行原地 (XIP) 子系统。缓存为提升性能分成两个存储区，但表现为单一16 kB缓存。XIP访问优先查询缓存。若未命中缓存，QMI将发起外部串行访问，将获取的数据加入缓存，并转发至系统总线（读取时）或与AHB写入数据合并（写入时）。



从闪存启动时，RP2350只读存储器启动程序（第5章）会设置基础的QMI执行原地配置。用户代码随后可重新配置此设置，以提升特定闪存设备的性能。QSPI时钟分频器可在任何时候更改，包括执行XIP时；其他重配置则需暂时禁用接口。

#### 4.4.1. XIP缓存

缓存容量为16 kB，2路组相联，命中延迟1个周期。该缓存位于XIP子系统内部，仅涉及QSPI存储接口访问，除非执行闪存编程操作，软件无需考虑缓存一致性。它缓存对26位下游XIP地址空间的访问。在RP2350上，该地址空间的下半部分被两个16 MB窗口占据，分别对应两个QMI芯片选择信号。RP2350为未来扩展保留了其余空间，但您可以使用该空间将缓存行固定在QMI地址空间之外，用作缓存即SRAM（见第4.4.1.3节）。在RP2350地址空间中，26位XIP地址空间多次镜像，并在系统总线地址的27:26位上进行解码：

- `0x10...` : 缓存的XIP访问
- `0x14...` : 非缓存的XIP访问
- `0x18...` : 缓存维护写入
- `0x1c...` : 非缓存且未转换的XIP访问 —— 绕过QMI地址转换

您可以通过`CTRL.EN_SECURE`和`CTRL.EN_NONSECURE`寄存器位，分别禁用安全和非安全访问的缓存查询。`CTRL`寄存器包含用于禁用对未缓存及未缓存/未翻译 XIP 窗口的安全/非安全访问的控制，从而避免了可能需要额外 SAU 或 PMP 区域的重复映射。

#### 4.4.1.1 缓存维护

缓存维护通过向 XIP 地址空间的缓存维护镜像逐行写入数据进行，起始地址为 `0x18000000`。缓存行大小为 8 字节。写入的数据会被忽略；地址的最低 3 位用于选择维护操作：

- `0x0`：按组/路使缓存失效
- `0x1`：按组/路清理缓存
- `0x2`：按地址使缓存失效
- `0x3`：按地址清理缓存
- `0x7`：固定地址处的缓存组/路（见第 4.4.1.3 节）

##### 使失效

将缓存行标记为不再包含有效数据；下一次访问相同地址时将导致缓存未命中。

不向外部存储器写回任何数据。用于外部存储器已被修改但缓存不会自动检测到此修改的情况，例如闪存编程操作。

##### 清除

指示缓存将任何由于之前缓存写访问而存储但尚未写回外部存储器的数据写出。用于使缓存写入的数据可被非缓存读取访问。

还用于缓存内容即将丢失但外部存储器仍保持供电时（例如系统即将断电）。

##### 按组/路

从构成缓存的 $2048 \times 8$ 字节行中选择特定缓存行进行维护。系统总线地址的第13位选择缓存路，第12至第3位选择该路中的特定缓存行。主要用于对所有缓存行进行遍历（例如在完全刷新缓存时使用）。

##### 按地址

在缓存中查找地址，若该缓存行当前已分配，则执行所请求的维护操作。仅当需要维护特定范围的XIP地址时使用，例如刚刚编程的闪存页。通常比完全刷新更快，因为缓存刷新的主要开销不在于维护操作，而在随之而来的大量缓存未命中。

##### 引脚

防止特定缓存行被驱逐。用于标记必须保证缓存命中的重要外部存储内容，或分配缓存行用作cache-as-SRAM。如果访问其他地址的缓存未命中且尝试驱逐被固定的缓存行，驱逐将失败，访问会降级为无缓存访问。

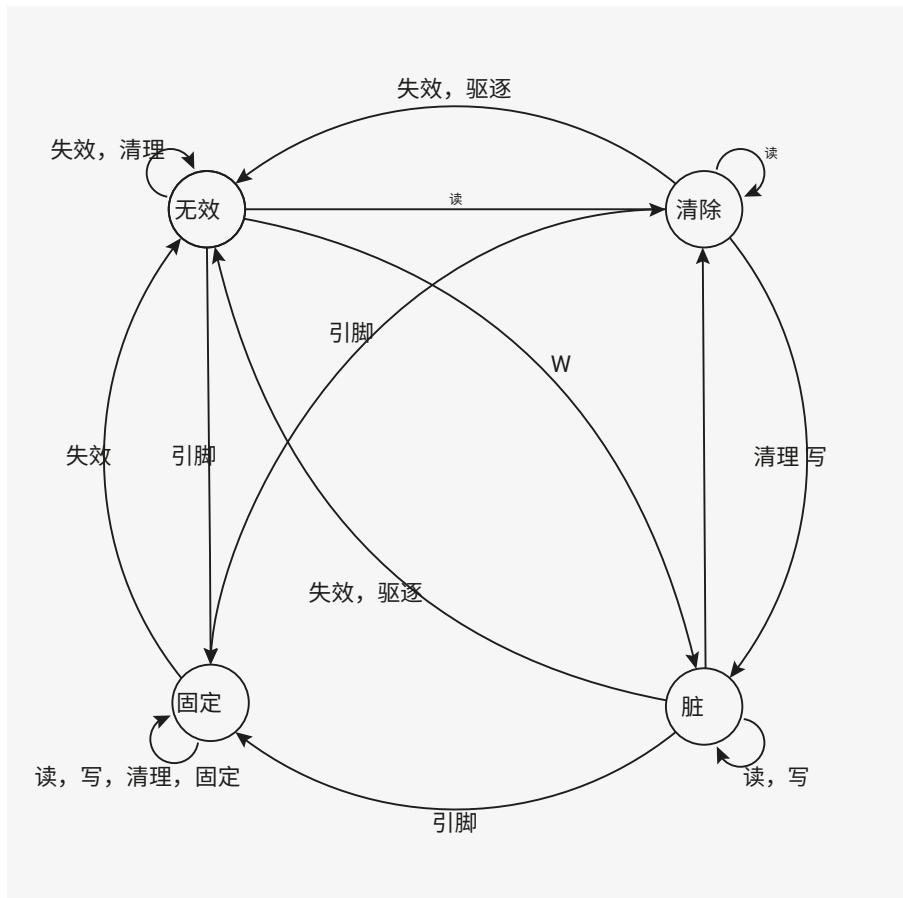
缓存维护操作作用于缓存的**tag memory**。这是缓存的元数据存储区，用于跟踪每条缓存行的状态。维护操作不影响缓存的**data memory**，后者包含来自外部存储的数据字节副本。

默认情况下，缓存维护仅限于安全模式。对缓存维护地址窗口的非安全写入无效且会返回总线错误。可以通过设置 CTRL\_MAINT\_NONSEC 寄存器位启用非安全缓存维护，但若安全软件可能执行缓存的 XIP 访问，则不建议启用该设置。

#### 4.4.1.2. 缓存行状态

缓存访问及维护操作所引起的缓存行状态变化，可归纳为一组状态转换。

图 17。各缓存行的状态转换示意图。失效、清理和固定分别表示失效、清理和固定的维护操作。读和写分别表示缓存的读取和写入操作。Evict 表示因读/写缓存未命中，为新的分配腾出空间而进行的缓存行去分配操作。



初始时，所有缓存行的状态均为未定义。从闪存启动时，bootrom 对缓存的每一行执行基于集合/路经的失效操作，以强制其进入已知状态。如上图所示，所有状态均有一个 Inv 弧指向无效状态。

一个 **dirty** 缓存行包含尚未写回至下游存储器的数据。

一个 **clean** 缓存行包含与下游存储器内容一致的数据。

访问无效缓存行会导致一次 **allocation**：缓存从下游存储器提取对应数据，存储于缓存中，然后将缓存行标记为 clean 或 dirty。缓存还存储部分下游地址，称为 **tag**，用以标识各缓存行对应的下游地址。读分配进入 clean 状态，因此缓存行可在任何时刻安全释放。写分配使缓存行进入脏状态，因此缓存行必须在释放之前向下游传播。

对于干净缓存行的写入会将其标记为脏状态，因为缓存中包含尚未向下游传播的写入数据。可以通过清理维护操作（`0x1`或`0x3`）显式将缓存行恢复为干净状态，但这并非必需。通常，当缓存需要重新分配缓存行时，会自动向下游传播脏缓存行。

驱逐发生在缓存的读写操作需要分配已处于干净或脏状态的缓存行时。

驱逐会使缓存行瞬间转为无效状态，以待分配。对于干净缓存行，此过程会瞬时完成。对于脏缓存行，缓存必须首先将其内容向下游传播，才能安全进入无效状态。

缓存行通过固定维护操作（`0x7`）进入固定状态，仅能通过失效维护操作（`0x0`或`0x2`）退出。

**i 注意**

固定操作仅将该行标记为已固定；它不会执行任何数据复制。当固定存在于外部存储设备中的行时，必须先固定该行，然后通过从未缓存的XIP窗口读取，将下游数据复制到该固定行中。

**4.4.1.3. 作为SRAM的缓存**

当您禁用RP2040的缓存时，缓存会映射整个缓存内存于 **0x15000000** 地址。RP2350以固定单个缓存行的能力替代了此功能。您可以通过以下方式使用此功能：

- 固定某地址范围内的整个缓存，将整个缓存用作SRAM
- 固定一条完整的缓存路，使缓存的一半可用作SRAM（剩余的缓存路仍按常规工作）
- 固定映射关键闪存内容的地址范围

**i 注意**

当通过CTRL寄存器（根据总线访问的安全级别为CTRL.EN\_SECURE或CTRL.EN\_NONSECURE）禁用缓存时，固定的缓存行将无法访问。

由于QMI仅占用64 MB XIP地址空间的下半部分，您可以将缓存行固定在QMI地址范围之外（例如XIP空间顶部），以避免干扰任何QMI访问。一般而言，固定的缓存行越多，其他访问的缓存命中率越低。

缓存行通过固定维护操作（**0x7**）进行固定，该操作包括以下步骤：

1. 使用维护操作的完整地址执行隐式的按地址失效操作（**0x2**）
  - 确保每个地址仅分配到一个缓存路（这是正确缓存操作的前提）
2. 选择要固定的缓存行，使用位 **13**选择缓存路，位 **12:3**选择缓存组（与 **0x0/0x1**按组/路失效或清理命令相同）
3. 将地址写入缓存行的标签项
4. 将缓存行的状态更改为固定状态（参见第4.4.1.2节的状态图）
5. 使用维护操作的完整地址更新缓存行的标签

执行固定操作后，对指定地址的缓存读写始终命中缓存，直到该缓存行被失效或重新固定到其他地址。

**i 注意**

固定两个模缓存大小相等的地址会导致同一缓存行被固定两次。它不会固定两条不同的缓存行。第二次固定会覆盖第一次。

当缓存访问命中固定的缓存行时，其行为与脏缓存行相同。缓存按常规方式读写，仿佛该缓存行已被正常分配。

缓存替换策略为随机，且仅尝试一次选择替换路径。如果缓存选择替换一个固定的缓存行，替换失败，该访问降级为非缓存访问。因此，具有一路固定缓存的缓存行为与直接映射的8kB缓存不完全相同，但平均性能相似。

缓存行状态存储于位于XIP存储器电源域内的缓存标签存储器中。该存储器内容在复位时不会更改，因此固定的缓存行在复位后仍保持固定状态。若XIP存储器电源域未关闭，内存内容在切换核心复位域的电源循环期间保持不变。引导只读存储器在进入闪存引导或NSBOOT（USB引导）路径时会清除标签存储器，但看门狗缓存向量重启可直接引导至固定的XIP缓存行。

#### 4.4.2. QSPI存储器接口（QMI）

未缓存访问及缓存未命中需要访问外部存储器。QSPI存储器接口（QMI）提供此访问，详见第12.14节。QMI支持：

- 最多两个外部QSPI设备，具独立芯片选择及共享时钟/数据信号
  - 分组配置寄存器，含不同的 SCK频率及QSPI操作码
- 内存映射读写操作（写操作须通过CTRL.WRITABLE\_M0/CTRL.WRITABLE\_M1启用）
- 串行/双路/四路SPI传输格式
- SCK速度最高可达 `clk_sys`
- 对未缓存访问支持8/16/32位访问，对缓存行填充支持64位访问
- 顺序寻址访问可自动链式合并为单次QSPI传输
- 地址转换（每个QSPI设备支持4x4 MB窗口）
  - Flash存储地址可能与运行时地址不同，例如适用于多个OTA升级映像槽
  - 允许代码和数据段，或安全与非安全映像分别映射
- 用于编程和配置外部QSPI设备的直接模式FIFO接口

通过两个缓存AHB端口及DMA流硬件进行的XIP访问，对QMI访问权进行仲裁。QMI由单独的APB端口配置。

QMI是为RP2350设计的新型存储器接口，替代了RP2040上的SSI外设。

#### 4.4.3. 流式DMA接口

由于Flash容量通常远大于片上SRAM，将数据块从Flash流式传输至内存通常非常有用。在前台软件执行其他任务的同时，让DMA在后台传输这些数据是十分便利的。如果代码能在此过程中继续从闪存执行，则更加方便。

这与标准XIP操作不兼容，因为QMI串行传输会导致DMA出现长时间总线阻塞。

对于处理器而言，这种阻塞是可接受的，因为有序处理器在等待指令获取完成期间通常无其他任务可做，且典型代码执行相比于大量流式传输不常访问的数据，具有更高的缓存命中率。相反，DMA的阻塞会阻止其他活动的DMA通道在此期间继续运行，从而降低整体DMA吞吐量。

STREAM\_ADDR和STREAM\_CTR寄存器用于配置一系列线性闪存读取操作。XIP子系统以尽最大努力的方式在后台执行这些读取。为了最大限度减少流式传输期间对从闪存执行代码的影响，流式传输硬件对QMI的访问优先级低于常规XIP访问，并且在最后一次XIP缓存未命中与恢复流式传输之间设有短暂冷却时间（9个周期）。这避免了在XIP缓存未命中时初始访问延迟的增加。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/flash/xip\\_stream/flash\\_xip\\_stream.c](https://github.com/raspberrypi/pico-examples/blob/master/flash/xip_stream/flash_xip_stream.c) 第45-48行

```

45 while (!(xip_ctrl_hw->stat & XIP_STAT_FIFO_EMPTY))
46 (void) xip_ctrl_hw->stream_fifo;
47 xip_ctrl_hw->stream_addr = (uint32_t) &random_test_data[0];
48 xip_ctrl_hw->stream_ctr = count_of(random_test_data);

```

流式数据被推入一个小型FIFO，该FIFO生成DREQ信号，指示DMA收集流式数据。由于DMA在从闪存读取数据后才开始发起读取，因此在访问数据时DMA不会发生阻塞。随后，DMA可通过辅助AHB端口检索这些数据，该端口提供对流式数据FIFO的单周期直接访问。

在RP2350上，您还可以使用辅助AHB端口访问QMI直接模式FIFO。这种访问方式比通过QMI APB配置端口访问FIFO更为高效。

通过QMI APB配置端口访问FIFO。当启用QMI访问链时，流式XIP DMA表现接近理论最大QSPI吞吐量，但在需求达到理论最大吞吐量的场景中，直接模式FIFO可通过AHB端口访问。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/flash/xip\\_stream/flash\\_xip\\_stream.c](https://github.com/raspberrypi/pico-examples/blob/master/flash/xip_stream/flash_xip_stream.c) 第58至70行

```

58 const uint dma_chan = 0;
59 dma_channel_config cfg = dma_channel_get_default_config(dma_chan);
60 channel_config_set_read_increment(&cfg, false);
61 channel_config_set_write_increment(&cfg, true);
62 channel_config_set_dreq(&cfg, DREQ_XIP_STREAM);
63 dma_channel_configure(
64 dma_chan,
65 &cfg,
66 (void *) buf, // 写入地址
67 (const void *) XIP_AUX_BASE, // 读取地址
68 count_of(random_test_data), // 传输计数
69 true // 立即开始!
70);

```

#### 4.4.4. 性能计数器

XIP 子系统提供两个性能计数器。它们为 32 位，达到 `0x` 时饱和值`xffffffff`，写入任意值可清零。计数内容包括：

1. XIP 访问的总次数，针对任何别名地址

2. 导致缓存命中的 XIP 访问次数

这为常用场景下缓存命中率的性能分析提供了一种方法。

#### 4.4.5. XIP\_CTRL寄存器列表

XIP 控制寄存器起始地址为 `0x400c8000`（在 SDK 中定义为 `XIP_CTRL_BASE`）。

表 439. XIP  
寄存器列表

| 偏移量  | 名称                       | 说明                 |
|------|--------------------------|--------------------|
| 0x00 | <code>CTRL</code>        | 缓存控制寄存器。非安全上下文下只读。 |
| 0x08 | <code>STAT</code>        |                    |
| 0x0c | <code>CTR_HIT</code>     | 缓存命中计数器            |
| 0x10 | <code>CTR_ACC</code>     | 缓存访问计数器            |
| 0x14 | <code>STREAM_ADDR</code> | FIFO 流地址           |
| 0x18 | <code>STREAM_CTR</code>  | FIFO 流控制           |
| 0x1c | <code>STREAM_FIFO</code> | FIFO 流数据           |

#### XIP: CTRL 寄存器

偏移: 0x00

##### 说明

缓存控制寄存器。非安全上下文下只读。

表 440. CTRL  
寄存器

| 位     | 描述                                                                                                                                                                                                                                                                                                                                                                                                                                               | 类型 | 复位  |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 31:12 | 保留。                                                                                                                                                                                                                                                                                                                                                                                                                                              | -  | -   |
| 11    | <p><b>WRITABLE_M1</b>: 若为1, 启用对 XIP 内存窗口1 (地址范围0x11000000至0x11fffff及其非缓存镜像) 的写操作。若为0, 则该区域为只读。</p> <p>XIP 内存默认处于只读状态。若将 RAM 设备连接至 QSPI 片选1, 必须设置此位以启用写入功能。</p> <p>默认的只读行为可避免向只读 QSPI 设备 (如闪存) 写入时出现的两类问题。首先, 写入操作因缓存机制初步呈现成功, 但当写入行被逐出时数据将丢失, 导致行为不可预测。</p> <p>其次, 当一行写操作被驱逐时, 将触发对闪存的写入命令, 这可能导致闪存退出连续读取模式。</p> <p>此后, 闪存的读取将返回无效数据。这是一个安全隐患, 因为若非安全软件被授权写入任何闪存地址, 将可能破坏安全闪存的读取过程。</p> <p>请注意, 只读行为是通过将写操作降级为读操作实现的, 因此写入仍会分配地址, 但不会产生其他影响。</p> | 读写 | 0x0 |
| 10    | <p><b>WRITABLE_M0</b>: 若设置为1, 允许写入XIP内存窗口0 (地址范围0x1至0x10ffff及其非缓存镜像)。0000000 若设置为0, 该区域则为只读。</p> <p>XIP 内存默认处于只读状态。当在QSPI芯片选择0上连接RAM设备时, 必须设置此位以启用写入权限。</p> <p>默认的只读行为可避免向只读 QSPI 设备 (如闪存) 写入时出现的两类问题。首先, 写入操作因缓存机制初步呈现成功, 但当写入行被逐出时数据将丢失, 导致行为不可预测。</p> <p>其次, 当一行写操作被驱逐时, 将触发对闪存的写入命令, 这可能导致闪存退出连续读取模式。</p> <p>此后, 闪存的读取将返回无效数据。这是一个安全隐患, 因为若非安全软件被授权写入任何闪存地址, 将可能破坏安全闪存的读取过程。</p> <p>请注意, 只读行为是通过将写操作降级为读操作实现的, 因此写入仍会分配地址, 但不会产生其他影响。</p>    | 读写 | 0x0 |
| 9     | <p><b>SPLIT_WAYS</b>: 若设置为1, 所有缓存且安全的访问将路由至缓存的way 0, 所有缓存但非安全的访问将路由至缓存的way 1。</p> <p>此操作将缓存划分为两个大小减半的直接映射区域, 以便非安全代码无法观察由安全执行引起的缓存行状态变化。</p> <p>更改 SPLIT_WAYS 的值时, 必须执行完整的缓存刷新。刷新应在 SPLIT_WAYS 为 0 时进行, 以确保两个缓存路均可访问并进行失效处理。</p>                                                                                                                                                                                                                 | 读写 | 0x0 |

| 位 | 描述                                                                                                                                                                                                                                                                | 类型 | 复位  |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 8 | <p><b>MAINT_NONSEC:</b> 当其值为 0 时，对缓存维护地址窗口（addr[27] == 1, addr[26] == 0）的非安全访问将产生总线错误。当其值为 1 时，非安全访问可通过写入缓存维护地址窗口来执行缓存维护操作。</p> <p>缓存维护操作可能被用来通过不当失效缓存行破坏安全数据，或通过固定缓存行将安全内容映射到非安全区域。因此，除非安全代码未使用缓存，否则该位通常应设置为 0。</p> <p>在允许非安全代码执行维护操作之前，应当仔细清理缓存数据存储器和标签存储器。</p> | 读写 | 0x0 |
| 7 | <b>NO_UNTRANSLATED_NONSEC:</b> 当值为1时，对未缓存且未翻译窗口（地址位[27:26] == 3）的非安全访问将触发总线错误。                                                                                                                                                                                    | 读写 | 0x1 |
| 6 | <b>NO_UNTRANSLATED_SEC:</b> 当值为1时，对未缓存且未翻译窗口（地址位[27:26] == 3）的安全访问将触发总线错误。                                                                                                                                                                                        | 读写 | 0x0 |
| 5 | <p><b>NO_UNCACHED_NONSEC:</b> 当值为1时，对未缓存窗口（地址位[27:26] == 1）的非安全访问将触发总线错误。此操作可能减少保护闪存内容所需的SAU/MPU/PMP区域数目。</p> <p>请注意，此操作不会禁用对未缓存且未翻译窗口的访问——详见NO_UNTRANSLATED_SEC。</p>                                                                                             | 读写 | 0x0 |
| 4 | <p><b>NO_UNCACHED_SEC:</b> 当值为1时，对未缓存窗口（地址位[27:26] == 1）的安全访问将触发总线错误。此操作可能减少保护闪存内容所需的SAU/MPU/PMP区域数目。</p> <p>请注意，此操作不会禁用对未缓存且未翻译窗口的访问——详见NO_UNTRANSLATED_SEC。</p>                                                                                                 | 读写 | 0x0 |
| 3 | <b>POWER_DOWN:</b> 当值为1时，缓存存储器将被断电。它们保持状态，但无法被访问。这减少了静态功耗。向此位写入1会强制将CTRL_EN_SECURE和CTRL_EN_NONSECURE置为0，即断电状态下缓存无法启用。                                                                                                                                             | 读写 | 0x0 |
| 2 | 保留。                                                                                                                                                                                                                                                               | -  | -   |
| 1 | <p><b>EN_NONSECURE:</b> 置1时，启用非安全访问缓存。启用后，对缓存区（addr[26] == 0）内的非安全XIP访问将查询缓存，只有在请求数据不存在时才执行QSPI访问。禁用后，安全访问忽略缓存内容，始终访问QSPI接口。</p> <p>对未缓存区（addr[26] == 1）的访问，无论此位状态如何，均不查询缓存。</p>                                                                                  | 读写 | 0x1 |

| 位 | 描述                                                                                                                                                                                                                                           | 类型 | 复位  |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|-----|
| 0 | <p><b>EN_SECURE:</b> 置1时，启用安全访问缓存。启用后，对缓存区 (addr[26] == 0) 内的安全XIP访问将查询缓存，只有在请求数据不存在时才执行QSPI访问。禁用后，安全访问忽略缓存内容，始终访问QSPI接口。</p> <p>对未缓存区 (addr[26] == 1) 的访问，无论此位状态如何，均不查询缓存。</p> <p>不存在缓存作为SRAM地址窗口。缓存行通过单独固定分配以实现类似SRAM的用途，同时保持缓存处于启用状态。</p> | 读写 | 0x1 |

## XIP: STAT寄存器

偏移: 0x08

表441. STAT  
寄存器

| 位    | 描述                                                                             | 类型 | 复位  |
|------|--------------------------------------------------------------------------------|----|-----|
| 31:3 | 保留。                                                                            | -  | -   |
| 2    | <b>FIFO_FULL:</b> 当该位为1时，表示XIP流式传输FIFO已完全满。流式传输FIFO深度为2条目，因此满标志和空标志可用以判断其当前状态。 | 只读 | 0x0 |
| 1    | <b>FIFO_EMPTY:</b> 当该位为1时，表示XIP流式传输FIFO已完全空。                                   | 只读 | 0x1 |
| 0    | 保留。                                                                            | -  | -   |

## XIP: CTR\_HIT寄存器

偏移: 0x0c

### 描述

缓存命中计数器

表442. CTR\_HIT  
寄存器

| 位    | 描述                                                     | 类型 | 复位         |
|------|--------------------------------------------------------|----|------------|
| 31:0 | 32位饱和计数器，每次缓存命中时递增，即XIP访问直接从缓存数据中获得服务时。<br>写入任意值以清除计数。 | WC | 0x00000000 |

## XIP: CTR\_ACC寄存器

偏移: 0x10

### 描述

缓存访问计数器

表443. CTR\_ACC  
寄存器

| 位    | 描述                                                        | 类型 | 复位         |
|------|-----------------------------------------------------------|----|------------|
| 31:0 | 一个32位饱和计数器，在每次XIP访问时递增，无论缓存是否命中，包括不可缓存的访问。<br>写入任意值以清除计数。 | WC | 0x00000000 |

## XIP: STREAM\_ADDR 寄存器

偏移: 0x14

**描述**

FIFO 流地址

表 444  
STREAM\_ADDR 寄存器

| 位    | 描述                                                                 | 类型 | 复位         |
|------|--------------------------------------------------------------------|----|------------|
| 31:2 | 从闪存流式传输到FIFO的下一个字地址。<br><br>每次闪存访问后自动递增。<br>在开始流式读取之前，将初始访问地址写入此处。 | 读写 | 0x00000000 |
| 1:0  | 保留。                                                                | -  | -          |

**XIP: STREAM\_CTR 寄存器**

偏移: 0x18

**描述**

FIFO 流控制

表 445  
STREAM\_CTR 寄存器

| 位     | 描述                                                                                                                                                        | 类型 | 复位       |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|----|----------|
| 31:22 | 保留。                                                                                                                                                       | -  | -        |
| 21:0  | 写入非零值以启动流式读取。该操作将在后台进行，利用闪存空闲周期将线性数据块从闪存传输到流式FIFO。<br><br>随着流的进行，自动递减（每次递减1），达到0时停止。<br>写入0可停止正在进行的流，丢弃任何正在传输的读取，以便立即启动新的流（需先清空FIFO并重新初始化STREAM_ADDRESS）。 | 读写 | 0x000000 |

**XIP: STREAM\_FIFO 寄存器**

偏移: 0x1c

**描述**

FIFO 流数据

表 446。  
STREAM\_FIFO 寄存器

| 位    | 描述                                                                 | 类型 | 复位         |
|------|--------------------------------------------------------------------|----|------------|
| 31:0 | 流式数据缓存在此处，供系统DMA读取。<br>该FIFO亦可通过XIP_AUX从属访问，以避免DMA因其他XIP流量引发的总线阻塞。 | RF | 0x00000000 |

**4.4.6. XIP\_AUX 寄存器列表**

XIP\_AUX端口提供对流式FIFO及QMI直通模式FIFO的高速AHB访问，降低DMA访问这些FIFO的开销。

表 447。XIP\_AUX 寄存器列表

| 偏移量 | 名称            | 说明                                     |
|-----|---------------|----------------------------------------|
| 0x0 | STREAM        | 读取XIP流FIFO（快速总线访问XIP_CTRL_STREAM_FIFO） |
| 0x4 | QMI_DIRECT_TX | 写入QMI直通模式TX FIFO（快速总线访问QMI_DIRECT_TX）  |

| 偏移量 | 名称            | 说明                                    |
|-----|---------------|---------------------------------------|
| 0x8 | QMI_DIRECT_RX | 读取QMI直通模式RX FIFO（快速总线访问QMI_DIRECT_RX） |

## XIP\_AUX: STREAM寄存器

偏移: 0x0

表448。STREAM  
寄存器

| 位    | 描述                                     | 类型 | 复位         |
|------|----------------------------------------|----|------------|
| 31:0 | 读取XIP流FIFO（快速总线访问XIP_CTRL_STREAM_FIFO） | RF | 0x00000000 |

## XIP\_AUX: QMI\_DIRECT\_TX寄存器

偏移: 0x4

### 描述

写入QMI直通模式TX FIFO（快速总线访问QMI\_DIRECT\_TX）

表449。  
QMI\_DIRECT\_TX  
寄存器

| 位     | 描述                                                                                                                                                           | 类型 | 复位     |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|----|--------|
| 31:21 | 保留。                                                                                                                                                          | -  | -      |
| 20    | <b>NOPUSH:</b> 禁止对应此TX FIFO条目的RX FIFO推送操作。<br><br>用于避免在SPI传输开始时推送命令时，RX FIFO中出现无效数据。                                                                         | WF | 0x0    |
| 19    | <b>OE:</b> 输出使能（高电平有效）。对于单宽度（SPI），此字段被忽略，SD0始终设置为输出，SD1始终设置为输入。<br><br>对于双宽和四宽（DSPI/QSPI），该字段设置传输此FIFO记录时相关SDx引脚是否设为输出。在此情况下，命令/地址应设置OE，数据传输则应根据传输方向设置或清除OE。 | WF | 0x0    |
| 18    | <b>DWIDTH:</b> 数据宽度。若为0，硬件传输DIRECT_TX DATA字段的8位最低有效位，并在DIRECT_RX中的8位最低有效位返回8位数据；若为1，使用完整16位宽度。8位和16位传输可自由混合。                                                 | WF | 0x0    |
| 17:16 | <b>IWIDTH:</b> 配置该FIFO记录是否以单、双或四接口宽度（0/1/2）传输。不同宽度可自由混合使用。                                                                                                   | WF | 0x0    |
|       | 枚举值：                                                                                                                                                         |    |        |
|       | 0x0 → S：单宽度                                                                                                                                                  |    |        |
|       | 0x1 → D：双宽度                                                                                                                                                  |    |        |
|       | 0x2 → Q：四宽度                                                                                                                                                  |    |        |
| 15:0  | <b>DATA:</b> 推入此处的数据将在SCK下降沿（若为首次脉冲，则在SCK第一个上升沿之前）时钟输出。对于每个输出的字节，接口将在SCK上升沿同时采样一个字节，并将其推送至DIRECT_RX FIFO。<br><br>对于16位数据，先传输最低有效字节。                          | WF | 0x0000 |

## XIP\_AUX: QMI\_DIRECT\_RX 寄存器

偏移量: 0x8

**描述**

读取QMI直通模式RX FIFO（快速总线访问QMI\_DIRECT\_RX）

表450。  
QMI\_DIRECT\_RX  
寄存器

| 位     | 描述                                                                                                                                  | 类型 | 复位     |
|-------|-------------------------------------------------------------------------------------------------------------------------------------|----|--------|
| 31:16 | 保留。                                                                                                                                 | -  | -      |
| 15:0  | <p>串行接口每输出一个字节的同时，将同步时钟输入一个字节，该字节存入此FIFO。当FIFO满时，串行接口将暂停，以避免数据丢失。</p> <p>当16位数据写入TX FIFO时，相应的RX FIFO中也将写入16位数据。最低有效字节是第一个接收的字节。</p> | RF | 0x0000 |

## 4.5. OTP

RP2350包含8 kB一次性可编程存储器（OTP），其存储内容包括：

- 制造信息，例如唯一设备ID
- 启动配置，例如非默认晶体振荡器频率
- 用于启动签名强制执行的公钥指纹
- 用于将外部闪存内容解密至SRAM的对称密钥
- 用户自定义内容，包括可启动程序映像（第5.10.7节）

OTP存储结构为 $4096 \times 24$ 位的行。每行包含16位数据和8位奇偶校验信息，提供8 kB数据存储容量。OTP位单元初始值为0，可被编程为1。但在任何情况下均无法将其清除回0。这确保了安全关键标志（如禁用调试）一旦设置后在物理上不可清除。但您也必须谨慎编程正确的值。

有关OTP子系统的详细信息，请参见第13章。

# 第5章 Bootrom

每个RP2350设备包含32 kB的掩模只读存储器（mask ROM）：一种物理不可变的存储资源，详见第4.1节。RP2350 bootrom是刻录于该只读存储器中的二进制映像，包含复位时执行的第一条指令。

bootrom概念部分（第5.1节）涵盖以下主题，是理解bootrom功能及其实现的必要基础：

- 分区表及其相关的闪存权限
- 可启动映像及存储其元数据的块循环
- 映像和分区表的版本控制，以及支持双缓冲升级的A/B版本
- 用于OTP中公钥指纹的安全启动的哈希和签名（另见安全章节第10.1.1节）
- 可启动映像的加载映射，以及bootrom根据加载映射从闪存加载至RAM的打包二进制文件
- 防回滚保护，用于撤销较旧且已被破坏的软件版本
- 闪存启动的三种形式：
  - 闪存镜像启动，即单个二进制镜像直接写入闪存
  - 闪存分区启动，从分区表中选择启动镜像
  - 分区表内嵌镜像启动，启动镜像不包含分区表，但嵌入了分区表数据结构以划分闪存地址空间
- 针对分区表A/B版本的启动槽
- 闪存更新启动，一种特殊的一次性启动模式，支持下载镜像后的版本回退
- 支持“先测试后升级”的分阶段升级以及镜像自检
- 闪存镜像地址转换，确保镜像的运行时地址与物理存储位置无关
- 尝试在Arm上运行RISC-V二进制文件或反之自动切换架构
- 根据权限和UF2 family ID，将UF2下载定向到不同的闪存分区

除上述功能外，RP2350只读存储器还实现了：

- 核心0的初始启动序列（第5.2节）
- 核心1低功耗等待与启动协议（第5.3节）
- 通过只读存储器符号表导出的运行时API（第5.4节），例如flash和OTP编程
- 可供非安全代码使用的运行时API子集，每个API入口点的权限由安全代码单独配置
- 支持UF2的符合USB MSC类的引导加载程序，用于向flash或RAM下载代码/数据（第5.5节），包括版本控制和A/B分区支持
- 用于高级操作如OTP编程的USB PICOBLOCK接口（第5.6节），并支持 `picotool` 或其他主机端工具
- 支持对白标记所有通过USB公开的信息/标识符（第5.7节）
- 提供最小化shell的UART引导加载程序，用于从主控微控制器加载SRAM二进制文件（第5.8节）

在深入了解上述功能列表之前，您应先阅读启动只读存储器概念章节。RP2350相较于RP2040启动只读存储器添加了大量新功能。如时间紧迫，第5.9.5节涵盖了在未启用安全启动情况下，二进制文件在RP2350上可启动的最低要求。

**Bootrom 源代码**

RP2350 bootrom 的所有源文件均依据三条款 BSD 许可证条款提供：

[github.com/raspberrypi/pico-bootrom-rp2350](https://github.com/raspberrypi/pico-bootrom-rp2350)

## 5.1. 启动只读存储器概念

下文各节中的粗体字介绍了一个概念。本章节频繁引用这些概念。

### 5.1.1. 安全与非安全

本数据手册采用（大写）术语 **Secure** 和 **Non-secure**，指代 Armv8-M 架构参考手册中定义的同名 Arm 执行状态。非大写的“secure”无特殊含义。

在某些情况下，Secure 也可指代 RISC-V 核心，通常是运行于机器特权级别的核。例如，低级闪存 API 仅向 Arm Secure 代码和 RISC-V 代码导出，因此 Secure 作为此类 API 的简写。

所谓**secured RP2350**是指启用了安全引导（第 5.10.1 节）的设备。此状态不同于 Secure 状态，因为设备在完成安全引导后，可能会运行 Secure 和 Non-secure 代码的混合。

### 5.1.2. 分区表

分区表将闪存划分为最多16个独立区域，称为分区。每个分区定义连续闪存地址范围的闪存权限等属性。**PARTITION\_TABLE** 数据结构描述分区表，是块的一个示例。分区表的使用完全是可选的。

将闪存划分为多个分区，可以使您：

- 在设备上存储多个可执行映像。例如：
  - 用于A/B启动版本（第5.1.7节）
  - 用于不同架构（Arm/RISC-V）或安全/非安全模式
  - 用于自定义引导加载程序
- 为数据预留空间。例如：
  - 嵌入式文件系统
  - 共享Wi-Fi固件
  - 应用程序资源
- 为闪存不同区域提供不同的安全属性（第5.1.3节）
- 根据系列ID将UF2下载目标定向至不同分区（第5.1.18节），包括针对您平台自定义定义的UF2系列

有关闪存启动期间**PARTITION\_TABLE**发现的更多信息，请参见第5.1.5.2节。

分区表可进行版本控制以支持A/B升级。分区表还可被哈希和签名，以保障安全性和完整性。我们建议对分区表进行哈希处理，以确保其未被损坏。当使用启动槽更新分区表时，这一点尤为重要，因为损坏的高版本分区表可能会被选择，而非低版本但未损坏的分区表。

### 5.1.2.1. 分区属性

每个分区指定其所涵盖闪存地址的分区属性，包括：

- 两个闪存窗口内32 MB逻辑地址空间的起始/结束偏移量；偏移量以闪存扇区（4 kB）的倍数表示。
- 可启动分区必须完全位于第一个16 MB闪存窗口内，因地址转换硬件的限制。
- 分区的访问权限：针对安全（**S**）、非安全（**NS**）和引导加载程序（**BL**）访问的读写权限。
- 通过UF2引导加载程序可写入该分区的UF2系列ID信息。
- 一个可选的64位标识符
- 一个可选名称（用于人类可识别的字符串）
- 是否在Arm或RISC-V启动过程中忽略该分区
- 用于将分区分组的信息（参见第5.1.7节及第5.1.18节）

[第5.9.4节](#)载明了完整的分区属性列表，及[PARTITION\\_TABLE二进制格式](#)。

若无分区表，则整个闪存视为单一区域，且无权限限制。无分区表时，不支持自定义UF2系列ID，必须使用表455中指定的标准ID之一。

### 5.1.3. 闪存权限

分区表（详见第5.1.2节）的一项作用是定义**闪存权限**，简称权限。

分区表为每个分区保存一组权限标志：单个分区覆盖的所有字节权限一致。分区表单独定义了**未分区空间**的权限，即不匹配任何分区表定义分区的闪存地址。

分别为安全区（**S**）、非安全区（**NS**）和引导加载程序（**BL**）访问设置读写权限。

引导加载程序权限控制UF2文件的写入位置，以及设备处于BOOTSEL模式时通过 **picotool** 可访问的内容。

由于闪存权限可能在运行时动态更改，部分分区表在运行时驻留于RAM中。您可以在运行时修改此表，为闪存的其他区域添加权限，而无需更改存储于闪存中的分区表本身。虽然没有bootrom API支持此功能，但内存中的分区表格式已被记录，且ROM表中提供了相关指针。SDK提供了封装此功能的API接口。

### 5.1.4. 镜像定义

镜像（**image**）是连续的数据块，可能包含代码、数据或两者的混合。镜像定义（**image definition**）是嵌入在镜像起始位置附近的元数据块。镜像定义中存储的元数据使bootrom能够识别有效的可执行与非可执行镜像。**IMAGE\_DEF** 数据结构以二进制格式表示映像定义，是块的一个示例。

对于可执行映像，**IMAGE\_DEF** 可视为类似 **ELF** 头部，因为其可包含映像属性，例如架构/芯片、入口点、加载地址等。

所有 **IMAGE\_DEF** 均可包含版本信息，且可被哈希或签名。尽管 bootrom 仅直接引导可执行映像，但它为用户应用程序提供了从一个或多个分区中选择有效（可能已签名）数据映像的功能。

有效的 **IMAGE\_DEF** 存在使 bootrom 能够区分闪存中的有效应用与随机数据。因此，任何拟引导的可执行二进制文件中均必须包含有效的 **IMAGE\_DEF**。

有关 bootrom 如何发现 **IMAGE\_DEF** 的详细信息，请参见块循环章节。

有关 `IMAGE_DEF` 格式的详细说明，请参见第5.9.3节。

有关可引导镜像最低要求的说明，请参见第5.9.5节。

## 5.1.5. 块及块循环

### 5.1.5.1. 块

`IMAGE_DEF` 和 `PARTITION_TABLE` 均为块的示例。块是一种可识别且自检的数据结构，包含一个或多个不同的数据项。块中首个项的类型决定了整个块的类型。

块具有向前与向后兼容性；项类型今后不会以导致现有代码误解数据的方式进行更改。块的使用者（包括只读存储器）必须跳过块内当前标记为保留的项类型；遇到保留项类型时，不得导致块校验失败。

为了被视为有效块，必须具备以下特性：

- 它必须以4字节魔术头 `PICOBIN_BLOCK_MARKER_START` (`0xffffded3`) 开头
- 每个（可变大小）项的结束位置必须同时为另一个有效项的起始位置
- 最后一项必须具有类型 `PICOBIN_BLOCK_ITEM_2BS_LAST` 且指定区块的正确完整长度
- 它必须以4字节魔术尾部 `PICOBIN_BLOCK_MARKER_END` (`0xab123579`) 结束

魔术头部和尾部的值被选为在可执行的Arm和RISC-V代码中极不可能出现。有关区块格式的详细信息，请参见第5.9.1节。

在给定的内存或闪存区域（如分区）中，通过搜索该区域的前4KB（用于闪存启动）或整个区域（用于RAM/OTP镜像启动）以定位属于有效块环的有效区块。

目前，RP2350启动只读存储器仅使用 `IMAGE_DEF` 和 `PARTITION_TABLE` 两种区块类型，但区块格式预留了编码空间以支持未来扩展。

### 5.1.5.2. 块循环

块循环是一种块的循环链表（一个链环）。每个块都具有指向下一个块的相对指针，且最后一个块必须链接回第一个块。单个块可以通过相对指针为0链接回自身，从而形成块循环。

循环中的第一个块必须具有所有块中最低的地址。

块循环的目的有三点：

- 发现哪些块属于同一镜像，避免暴力搜索。
- 允许在后链接处理步骤中追加元数据。
- 检测二进制部分被覆盖，从而破坏循环的情况。

对于闪存镜像启动，bootrom会搜索闪存的前4 kB；4 kB大小是在兼顾不同语言内存布局灵活性与避免在尝试不同闪存访问模式和QSPI时钟频率时扫描过多闪存之间的折衷。闪存分区启动同样将搜索限制在分区的前4 kB范围内。

搜索窗口可能更大，例如在UF2 SRAM下载后引导RAM镜像时，搜索窗口涵盖整个SRAM。为实现最快启动时间，应尽量将第一个块定位于二进制文件的开头附近。

块循环支持多个块，原因如下：

- 签署镜像时，会复制现有的 `IMAGE_DEF`，并添加一个更大的 `IMAGE_DEF`，内含附加的签名信息。
- 一个镜像可能包含多个 `IMAGE_DEF`，例如使用不同的签名密钥。

- 将块同时置于镜像的开头和结尾，可检测镜像的部分覆盖（例如由过度热情使用绝对地址 UF2 下载导致的覆盖）。此功能为 SDK 默认设置。对整个镜像进行哈希或签名更为稳健，因为它能检测镜像中间的损坏。

- 通用二进制镜像可能包含 Arm 和 RISC-V 代码，包括两者的 `IMAGE_DEF`。
- 在嵌入式分区表情况下，`PARTITION_TABLE` 与 `IMAGE_DEF` 可同时存在于同一块循环中。

如果一个块循环包含多个 `IMAGE_DEF` 或多个 `PARTITION_TABLE`，通常最后一个被链表遍历到的将被视为胜出者。例外情况为针对不同架构（Arm 和 RISC-V）的两个 `IMAGE_DEF`；当前正在执行 bootrom 的架构对应的 `IMAGE_DEF` 始终优先于其他架构的 `IMAGE_DEF`。

### 5.1.6. 块版本控制

任何块都可以包含一个 `version`。版本信息由包含两个或三个16位值的元组组成：  
`(rollback).major.minor`，其中 `rollback` 部分为可选。类型为 `VERSION` 的条目包含定义块版本的二进制数据结构。

`rollback` 版本仅可为 `IMAGE_DEF` 指定，未指定时默认为零。您不能为分区表指定此版本。`rollback` 版本可用于安全的 RP2350，该版本与存储于 OTP 中的当前 `rollback` 版本号配合使用，能够防止在安装新版本后重新安装旧版本的易受攻击代码（参见第 5.1.11 节）。

完整版本号可用于在两个 `IMAGE_DEF` 或两个 `PARTITION_TABLE` 之间选择最新版本（详见第 5.1.7 节）。版本按字典序排序比较：

- 如版本 `x` 的回滚版本与版本 `y` 不同，则较大的回滚版本决定整体较高版本
- 如版本 `x` 的主版本与版本 `y` 不同，则较大的主版本决定整体较高版本
- 否则，次版本决定 `x` 与 `y` 中较高的版本

有关区块中 `VERSION` 项的详细信息，请参见第 5.9.2.1 节。

### 5.1.7. A/B 版本

一对分区可被组合为一个 **A/B** 对。通过逻辑分组 A 和 B 分区，您可将当前可执行映像（或数据）保存在一个分区中，并将新版本写入另一分区。写入新版本完成后，您可安全切换至该版本，如有问题可回退至旧版本。此操作可避免产生可能导致 RP2350 无法启动的部分写入状态。

- 启动 A/B 分区对时，启动只读存储器通常优先使用版本较高的分区。如遇非上述情况，详见第 5.1.16 节。
- 将 UF2 文件拖放至 BOOTSEL USB 驱动器时，UF2 针对的分区与启动时优先的 A/B 分区相反。详情请参见第 5.1.18 节。

#### ① 注意

分区表亦可具有 A/B 版本。有关该高级主题的更多信息，请参考第 5.1.15 节。

### 5.1.8. 哈希与签名

任意块均可被哈希或签名。哈希块存储映像哈希值（见第 5.9.2.3 节）。运行时，启动只读存储器计算哈希值并与存储的哈希进行比对，以验证该块的有效性。哈希用于防止

映像损坏，但不提供任何安全保障。

在受保护的 RP2350 上，仅凭哈希值不足以使映像被视为有效。所有映像必须具有签名：由私钥加密的哈希值，以及描述哈希生成过程的元数据（亦包含于哈希中）。此签名作为 **IMAGE\_DEF** 块的一部分进行存储。在 **IMAGE\_DEF** 块中包含签名的映像称为**签名映像**。

### 注意

有关签名和启动密钥的背景信息，请参见安全章节中安全启动的介绍（第10.1.1节）。

要验证签名映像，bootrom 使用 secp256k1公钥解密存储在签名中的哈希值；同时，bootrom 还会计算自身的映像哈希值，并将计算结果与签名中的哈希值进行比对。

公钥也通过 **SIGNATURE**项存储于区块中（参见第5.9.2.4节）：该密钥的（SHA-256）哈希必须与存储在 OTP 位置 BOOTKEY\_Y0\_0 及后续位置的启动密钥哈希之一相匹配。最多可在OTP中注册四个公钥，数量由BOOT\_FLAGS1.KEY\_VALID和BOOT\_FLAGS1.KEY\_INVALID确定。密钥的哈希值亦称为**密钥指纹**。

待哈希的数据由 **HASH\_DEF**项定义（详见第5.9.2.2节），该项指明哈希类型。此外，该项还指示应哈希的块本身的数据量。对于已签名块，哈希值必须包含直到最终 **SIGNATURE**项的所有块内容。

为确保有效，您的哈希或签名必须涵盖实际映像数据及块中存储的元数据。块的加载映射项指定bootrom在哈希或签名验证过程中需哈希的数据。

上述讨论主要适用于**IMAGE\_DEF**。在设置了`BOOT_FLAGS0.SECURE_PARTITION_TABLE`标志的安全型RP2350上，bootrom也强制对 `PARTITION_TABLE`进行签名验证。

## 5.1.9. 加载映射

加载映射描述二进制文件中的区域及在只读存储器启动程序执行二进制文件前应如何处理这些区域。

加载映射支持：

- 将二进制文件的部分内容从闪存复制到RAM（或XIP缓存）
- 清除部分RAM（包括 `.bss`段的清零，或在安全启动期间擦除未初始化内存）
- 定义二进制文件中哪些部分包含在哈希或签名中
- 防止刷新XIP缓存，以保持已加载的缓存行固定，直到二进制文件启动

有关 `LOAD_MAP`项类型在 **IMAGE\_DEF** 块中的详细说明，请参见第5.9.3.2节。

从闪存启动签名二进制文件时，应在验证签名并随后执行该二进制文件之前，将签名的数据和代码加载到RAM。否则，攻击者可能在签名验证和执行之间替换闪存设备，从而破坏验证过程。因此，加载映射还可作为哈希或签名所包含内容的便捷描述。加载映射本身受哈希或签名保护，且整个元数据块在处理前已加载至RAM，因此本身不涉及时序检查（time-of-check）与使用时检查（time-of-use）的问题。

## 5.1.10. 打包二进制文件

如第5.1.9节所述，受保护的RP2350上的签名二进制文件通常从闪存加载至RAM，在RAM中进行签名验证，随后从已验证的RAM版本执行。

打包二进制文件是指存储于闪存中、完全从RAM运行的二进制文件。该二进制文件通常编译为仅从RAM运行的RAM专用二进制文件（SDK中称为no\_flash），但随后经过闪存驻留的后处理。

驻留。启动只读存储器（bootrom）在执行前将二进制文件解包至RAM。

作为打包流程的一部分，诸如 `picotool` 的工具会添加 `LOAD_MAP`，指示启动只读存储器将闪存驻留映像的哪些部分加载至RAM及其存放位置。该工具可能在同一过程中对二进制文件进行哈希或签名。在此情况下，只读存储器在解包二进制文件时对所加载的数据进行哈希处理，以及对相关元数据（例如 `LOAD_MAP` 本身）进行哈希处理。只读存储器将生成的哈希值与 `IMAGE_DEF` 中预先计算的哈希值或签名进行比较，以验证解包后位于RAM中的内容，然后才运行该内容。

相比之下，RP2040中从RAM执行的闪存驻留二进制文件（SDK术语中的`copy_to_ram`二进制）必须先从闪存执行，然后将自身复制到RAM，再从RAM继续执行。在RP2040的情形中，加载程序自身（或更准确地说，SDK的 `crt0`）在闪存中就地执行以完成复制。这使得进行任何可信赖级别的验证变得不可能，因为加载程序本身是在不受信任的内存中执行。

### 5.1.11. 防回滚保护

RP2350上的防回滚功能可防止启动存在已知漏洞的旧版本二进制文件。即使该二进制文件已正确签名且满足启动的所有其他要求，防回滚功能仍将阻止其启动。

完整的 `IMAGE_DEF` 版本信息格式为 `(rollback).major.minor`，其中 `rollback` 部分为可选。若存在 `rollback` 版本，则附带一份OTP行列表，其有序值用于形成一个表示设备可运行的最小回退版本的比特 `thermometer`。

`thermometer` 编码是一种基数为1（单元制）的数字，其中整数值为最高位已置位比特的索引加一。例如，比特串 `00001111`、`00001001` 与 `00001000` 均编码为数值4，而全零比特模式编码为数值0。bootrom采用此编码的原因包括：

- 它允许包含计数器的OTP行递增，且
- 不允许其递减。

在受保护的RP2350上，bootrom将 `IMAGE_DEF` 的回退版本与存储在OTP中的 `thermometer` 编码的最小回退版本进行比较。若 `IMAGE_DEF` 值较低，bootrom将拒绝启动该映像。

`IMAGE_DEF` 的回滚版本受映像签名保护，故无法被不知签名密钥的对手修改。定义芯片最小回滚版本的OTP行列表亦存储于程序映像中，且同样受映像签名保护。

`IMAGE_DEF` 中的OTP行列表必须始终比 `IMAGE_DEF` 的回滚版本多至少一位（由 `picotool` 强制执行）。因此，旧版二进制文件始终包含足够信息，使 bootrom 能检测芯片的最小回滚版本已超出 `IMAGE_DEF` 中的回滚版本。在新版二进制文件中，您可向列表追加更多行，从而支持更高的回滚版本而无歧义。

当具有非零回滚版本的可执行映像成功启动时，其回滚版本将写入OTP温度计。`BOOT_FLAGS0.ROLLBACK_REQUIRED` 标志可用于要求 `IMAGE_DEF` 在受保护的 RP2350 上具有回滚版本。该标志在更新 OTP 中的回滚版本时自动设置。

#### 注意

带有回滚版本号为0的 `IMAGE_DEF` 不会自动设置 `BOOT_FLAGS0.ROLLBACK_REQUIRED` 标志，因此建议使用的最小回滚版本号为1，除非在预配置时手动设置了 `BOOT_FLAGS0.ROLLBACK_REQUIRED` 标志。

### 5.1.12. 闪存镜像启动

RP2350主要设计用于从QSPI闪存设备运行代码，该设备可为封装内或焊接至电路板。代码可直接在闪存中就地运行，或从闪存加载后在SRAM中执行。闪存启动指发现代码并准备运行的过程。闪存映像启动使用直接存储于闪存中的程序二进制文件，而非闪存分区中的文件。闪存映像启动要求启动只读存储器发现闪存前4 kB内开始的块循环，其中包含有效的 `IMAGE_DEF`（且不包含 `PARTITION_TABLE`）。

闪存映像启动无分区表，因此不能用于需要独立A/B分区的A/B版本校验。如果 **IMAGE\_DEF**有效（包括在受保护的RP2350上需要签名），则会启动该映像。

对于未签名的情况，**IMAGE\_DEF**的大小可小至20字节；详见第5.9.5节。

#### 💡 提示

此场景的更复杂版本是在块循环中存储多个 **IMAGE\_DEF**。在此情况下，如有效，则启动当前架构的最后一个**IMAGE\_DEF**。您可使用此方法为多种支持的架构实现通用二进制文件，或包含多个签名以面向不同密钥的设备。

### 5.1.13. 闪存分区启动

如果在从闪存前4 kB起始的有效块循环中找到**PARTITION\_TABLE**，但未找到 **IMAGE\_DEF**，且该**PARTITION\_TABLE**有效（包括在受保护的RP2350上需要签名），引导只读存储器将搜索该分区表内的分区以启动可执行映像。该过程成功时，称为flash partition boot。

分区按顺序搜索，跳过当前架构下标记为忽略的分区。只读存储器忽略分区是一种优化手段，也用于防止自动架构切换。

如果分区不是A/B配对的一部分，则在前4 kB内搜索有效块循环的起始位置。若发现有效块循环，且其中包含带有有效（包括在安全的RP2350上签名）**IMAGE\_DEF**的可执行映像，则选择该可执行映像引导。

若该分区为A/B配对中的A分区，只读存储器将按上述方式搜索两个分区。若两分区均包含可启动的 **IMAGE\_DEF**，则选择版本号较高的 **IMAGE\_DEF**。否则，选择有效的**IMAGE\_DEF**。该规则在高级场景中存在例外；详情请参见第5.1.16节及第5.1.17节。

### 5.1.14. 镜像内分区表启动

若在前4 kB闪存内启动的有效块循环中同时找到**PARTITION\_TABLE**和 **IMAGE\_DEF**块，则执行第三种类型的闪存引导。**IMAGE\_DEF**和**PARTITION\_TABLE**必须仅被识别，但不必是有效的或正确签名的。该规定旨在防止因果循环。

这称为**partition-table-in-image**启动，因为应用程序包含分区表（而非反之）。  
该分区表被称为**嵌入式分区表**。

**PARTITION\_TABLE**被加载为当前分区表，**IMAGE\_DEF**则被直接启动。**PARTITION\_TABLE**定义的表中不会搜索用于启动的**IMAGE\_DEF**。

以下常见情况可能采用此方案：

- 您仅使用**PARTITION\_TABLE**来管理闪存权限。您希望先加载该分区表，然后按正常方式启动。
  
- **IMAGE\_DEF**含有一个与分区表一同存放的小型引导程序。在此情况下，分区表将被再次加载，并进入相应映像。  
进入的映像随后很可能自分区表中选取分区，并从中启动映像。

### 5.1.15. 闪存启动槽

本章前述部分讨论了起始于首个4 kB闪存区内的块循环。该块循环包含了 **IMAGE\_DEF**、分区表（查找 **IMAGE\_DEF**）或 **IMAGE\_DEF**和**PARTITION\_TABLE**（未查找）。

上述所有情况均在**slot 0**中发现其块循环。在某些情况下，相邻的**slot 1**也会被搜索。

Slot 0起始于闪存起始处，大小为  $n \times 4\text{ kB}$  扇区。Slot 1大小相同，紧随slot 0之后。 $n$ 的默认值为1。两个slot均为4 kB大小，但可以通过指定一个值在 `FLASH_PARTITION_SLOT_SIZE` and 然后设置 `BOOT_FLAGS0.OVERRIDE_FLASH_PARTITION_SLOT_SIZE` 来覆盖此值。

类似于在A/B分区间选择 `IMAGE_DEF` 的方式，可以通过两个启动槽在A/B分区表之间进行选择。此功能支持对分区表进行版本控制，针对性地拖放包含分区表的UF2文件（参见第5.1.18节），类似于映像的处理流程。

槽1仅在有可能使用分区表时才有效。在简单情况下，如果在从槽0开始的块循环中检测到 `IMAGE_DEF` 且未检测到 `PARTITION_TABLE`，则该映像很可能实际上覆盖了槽1所在的空间，但由于没有 `PARTITION_TABLE`，槽1会被忽略。

如果槽0包含 `PARTITION_TABLE` 或不包含 `IMAGE_DEF`（包括槽0为空或含有无效数据），则可以考虑使用槽1。作为一种优化，前述情况下可通过在 `PARTITION_TABLE` 中设置单例标志来阻止对槽1的扫描。

### **① 注意**

当槽中同时存在 `IMAGE_DEF` 时，`PARTITION_TABLE` 中的 `VERSION` 字段决定使用槽0还是槽1。版本比较时会忽略 `IMAGE_DEF` 的元数据。

## 5.1.16. 闪存更新启动及版本降级

通常情况下，槽位0与槽位1的选择，以及分区A与分区B的选择，是基于各槽位或分区中有效的 `PARTITION_TABLE` 或 `IMAGE_DEF` 的版本来决定的。两个版本中较高的版本将优先采用。

然而，在使用 A/B 分区时，允许降级到较低版本的 `IMAGE_DEF`，前提是不违反受保护 RP2350 上的防回滚规则。

在此情况下，将新映像（及其 `IMAGE_DEF`）下载到当前非启动分区并执行正常重启将无效，因为新下载的映像版本较低。

为此，您可以通过看门狗擦写寄存器传递 `FLASH_UPDATE` 启动类型常量标志及指向刚更新的闪存区域起始位置的指针，启用 `flash update boot` 启动。

`bootrom` 在对写入 USB 大容量存储驱动器的闪存 UF2 进行编程后，会自动执行闪存更新启动。您也可以通过 `reboot()` API（参见第5.4.8.24节）以编程方式调用闪存更新启动程序。

通过重启参数传递的闪存地址范围在闪存更新启动过程中会被特殊处理。如区域起点为相应槽位或分区的起始位置，槽位中的 `PARTITION_TABLE` 或分区中的 `IMAGE_DEF` 将被选作启动，版本无关。

为确保保持降级持续，必须擦除之前启动槽位或分区的第一个扇区，以使新安装的 `PARTITION_TABLE` 或 `IMAGE_DEF` 在后续启动中持续被选用。该擦除操作于 `FLASH_UPDATE` 启动期间按以下方式执行。

1. 当 `PARTITION_TABLE` 有效（且如有需要已正确签名）且其槽位被选为启动时，另一个槽位的第一个扇区将被擦除。
2. 当有效的（且如有需要已正确签名） `IMAGE_DEF` 被启动时，另一个镜像的第一个扇区将被擦除。
3. 在镜像明确请求时，启动后将擦除另一个镜像的第一个扇区。这是前述标准行为的替代方案，通过 `IMAGE_DEF` 中的特殊“先试后买”标志进行选择。有关此功能的详细信息，请参见第5.1.17节。

### ⓘ 注意

在使用单槽或独立（非A/B）分区时，闪存更新和版本降级无效。

## 5.1.17. 试用功能

“先试后买”（缩写为TBYB）是一个仅限于IMAGE\_DEF的功能，允许进行完全安全的版本升级流程：

1. 可执行镜像当前正在，例如，分区B中运行。
2. 新的镜像被下载至分区A。
3. 下载完成后，将针对新更新的分区A执行一次FLASH\_UPDATE重启。
4. 只读存储器将优先尝试启动分区A（因闪存更新而设置）。请注意，在正常的非FLASH\_UPDATE启动过程中，非TBYB镜像在A/B分区中始终优先于TBYB镜像被选择。
  - 如果新映像未通过验证或签名，则后续的（非FLASH\_UPDATE）启动将使用分区B中的旧映像，以实现升级失败的恢复。
5. 如果新映像有效（且在必要时签名正确），则其将在看门狗定时器监控下运行，拥有16.7秒的时间通过explicit\_buy() 函数显式标记自身为OK。
  - 若映像完成回调，则擦除另一个分区（包含映像B）的首个4 KB扇区，且清除当前映像的TBYB标志，从而使A分区成为后续启动的首选分区。
  - 若映像未在指定时间内回调，系统将自动重启，并持续启动分区B（包含原始映像），因分区A仍标记为TBYB映像。

擦除A/B配对中对应分区的首个扇区，将切断其映像的块循环，使其无启动能力。此机制确保在TBYB下启动的临时映像即使版本低于对立映像，也会成为后续的首选启动映像。

看门狗超时时间固定为16.7秒（基于1微妙时间基准的24位计数）。进入目标映像后，该时间可以缩短，例如在自检例程仅需数百毫秒的情况下。也可以通过重新加载看门狗计数器来延长超时，但如果因故障导致反复重新加载看门狗，系统可能会停留在暂存映像中。

## 5.1.18. UF2 目标支持

第5.5节描述了USB大容量存储驱动器及其下载UF2文件功能，用于在RP2350上存储和/或执行代码或数据。

鉴于RP2350支持多处理器架构和多分区分区表，部分设备信息需用于确定如何处理基于闪存地址的UF2。根据具体情况，UF2中的闪存地址可能为绝对闪存存储地址（如RP2040一贯采用），也可能为闪存分区内的代码及数据的运行时地址。**UF2 targeting**指只读存储器引导程序（bootrom）用于解释UF2文件中闪存地址的规则。

UF2文件支持在文件中嵌入32位家族ID。该机制使设备能够识别专门针对其的固件，而非针对其他设备的固件。RP2350只读存储器引导程序识别表455中定义的标准UF2家族ID（rp2040、rp2350-arm-s、rp2350-arm-ns、rp2350-riscv、data及absolute）。您可在分区表中定义自有家族ID，以实现更精确的定位。

UF2家族ID的使用方式如下：

1. 带有absolute家族ID的UF2文件下载时不考虑分区边界。是否允许absolute家族ID下载，由分区表（如存在）或只读存储器配置（OTP）决定。默认出厂设置允许absolute家族ID的下载。

2. 如果不存在分区表，则默认允许 `data`、`rp2350-arm-s`（若启用Arm）和`rp2350-riscv`（若启用RISC-V）系列ID。UF2始终下载到闪存起始位置。
3. 若存在分区表，则非绝对系列ID定位于由分区表管理的单个分区：
  - a. UF2不会下载到无 `BL`写入闪存权限的分区。
  - b. 各分区会列出其接受的系列ID（包括RP2350标准与用户自定义ID）。
  - c. 对于A/B分区；A分区指示支持的系列ID，UF2将下载至非当前启动的分区（即若设备立即重启，此分区不会被选中）。
  - d. 允许对A/B分区进行进一步细分，以支持包含主分区使用（拥有）的数据/可执行文件的辅助A/B分区；详见第5.1.18.1节。

有关选择UF2目标分区时所采用的具体规则，详见第5.5.3节。

#### 注意

UF2系列ID用于在将UF2文件复制至USB驱动器时，或执行`picotool load -p`操作时进行分区定位。  
当使用`picotool load`命令且未使用`-p`标志时，映像可写入任何具有 `BL`写权限的闪存区域。

### 5.1.18.1 所有权分区

可执行文件可能需要来自其他分区的数据（如Wi-Fi固件）。当主可执行文件存储于A/B分区中时，为确保安全升级，可能需要将两额外分区C和D与主分区A和B关联，使得：

- 分区C中的数据供分区A中的可执行文件使用，
- 分区D中的数据供分区B中的可执行文件使用。

此情况下，分区表中标记A为C的所有者，C为A的**所有权分区**。这会影响UF2镜像的下载，由于其UF2家族ID，目标分区为C和D。

当UF2下载定位于C/D分区时，bootrom会检查A和B所属分区的状态，以决定受到控制的分区（C或D）接收下载。默认情况下：

- 如果B是具有A/B兼容家族ID的UF2的目标分区，则具有C/D兼容家族ID的UF2的目标分区为D。
- 反之，当A是A/B下载的目标分区时，C就是C/D下载的目标分区。

分区表中的`FLAGS_UF2_DOWNLOAD_AB_NON_BOOTABLE_OWNER_AFFINITY`标志会颠倒此映射关系。

### 5.1.19. 地址转换

RP2040要求镜像存储于闪存起始位置（`0x10000000`）。RP2350允许在分区中的任意位置存储可执行镜像，以支持通过A/B版本实现更为稳健的升级周期及其他用途。这导致可执行文件的链接地址及其二进制内容必须依赖于存储地址。该问题在一定程度上可通过位置无关代码规避，但会牺牲代码体积和性能。

RP2350通过硬件和bootrom对**地址转换**的支持，避免了此类问题。存储于闪存任意4 kB对齐位置的映像可映射至闪存地址`0x10000000`，且生效于运行时。SDK默认继续假设映像基址为`0x10000000`。

从分区启动映像时，bootrom会初始化QMI寄存器ATRANS0至ATRANS3，将闪存的运行时地址`0x10000000`（默认）映射至该分区的闪存存储地址起始处。映射区域的总大小被设置为分区大小，最大不得超过16 MB。访问超过启动分区大小的闪存地址（但低于`0x11000000`芯片选择水位标界限）时，将导致总线错误。

例如，若启动分区大小为 6 MB，且起始于闪存第 1 MB 处，则寄存器配置如下：

| 名称 / 内存起始地址                       | 闪存起始地址                  | 大小   |
|-----------------------------------|-------------------------|------|
| ATRANS0 / <code>0x10000000</code> | <code>0x10100000</code> | 4 MB |
| ATRANS1 / <code>0x10400000</code> | <code>0x10500000</code> | 2 MB |
| ATRANS2 / <code>0x10800000</code> | -                       | 0    |
| ATRANS3 / <code>0x10c00000</code> | -                       | 0    |

这将物理闪存范围从 `0x10100000` 至 `0x10700000` 映射至闪存地址 `0x10000000` 至 `0x10600000`，确保该分区在运行时出现在闪存起始位置。

在本例中，由于 ATRANS2 和 ATRANS3 不需要映射该分区，它们可由应用程序用于映射闪存的其他部分。

将分区起始映射到运行时地址 `0x10000000` 是默认行为，但您可以选择不同的地址，需遵循某些限制。Bootrom 允许运行时地址值为 `0x10000000`、`0x10400000`、`0x10800000`，以及 `0x10c00000` 作为映射区域的起始地址，具体选择由 `IMAGE_DEF` 指定。您必须将二进制文件链接至正确的、更高的基址以确保运行。例如，当应用程序从高闪存地址运行并保持映射，同时在地址 `0x1` 启动第二个应用程序时，此功能尤为有用。`0000000`。当安全映像为非安全客户端映像提供服务时，可能会采用此配置。

该自定义地址转换由负的 `ROLLING_WINDOW_DELTA` 值启用（详见第 5.9.3.5 节）。上述四个运行时地址对应的 `ROLLING_WINDOW_DELTA` 值分别为 `0`、`-0x400000`、`-0x800000` 或 `-0xc00000`，且仅支持这些非正值。该增量表示运行时地址 `0x1` 相对于分区起始位置的字节偏移量。`0000000` 出现。负的增量值表示地址 `0x10000000` 出现在分区起始地址之前；分区起始地址比 `0x1` 高出相应字节数。`0000000`。

正值同样有用，例如在作为后处理步骤时，将数据预置到已链接镜像的前端。正的增量必须是 4 KB 的倍数。例如，`ROLLING_WINDOW_DELTA` 为 `0x1000` 将设置地址转换，使分区内偏移 `0x1000` 处开始的镜像数据映射到 `0x10000000` 在运行时，映射区域中省略镜像的首个 4 KB。首个 4 KB 不可访问，除非通过未转换的 XIP 窗口，该窗口默认仅允许安全访问。

### 注意

由于 `0x10000000` → `0x11000000` 和 `0x11000000` → `0x12000000` 两个窗口内的地址转换是独立的，因此仅能从完全位于前 16 M B 闪存内的分区启动。

该地址转换由 QMI 中的硬件执行。详情请参见第 12.14.4 节。

### 5.1.20. 自动架构切换

如果 bootrom 遇到一个针对非当前架构的有效且正确签名的 `IMAGE_DEF`（如 Arm 模式启动时为 RISC-V，或 RISC-V 模式启动时为 Arm），则会执行一次 **自动架构切换**。bootrom 将重新启动至其发现的二进制文件对应的正确架构，该二进制文件随后在第二次尝试时成功启动。

传递至看门狗临时寄存器的信息（例如 RAM 映像的启动类型）将被保留，因此第二次启动时会作出与第一次相同的决策，最终选择相同的首选映像启动。

仅在以下情况发生此操作：

- 根据 OTP 关键标志，待切换的架构可用
- `BOOT_FLAGS0.DISABLE_AUTO_SWITCH_ARCH` 标志未禁用自动架构切换功能
- bootrom 在找到另一个架构的有效二进制文件之前，未发现针对当前架构的有效二进制文件

### 💡 提示

在闪存中存储适用于两种架构的可执行映像时，通常优先启动当前架构的映像。为此，应将映像保存在不同分区中，并在RISC-V架构下将Arm分区标记为忽略，反之亦然。这样可以避免始终选择第一个分区中的映像，并自动切换至另一架构下运行。

有关架构切换的硬件支持详细信息，请参见第3.9节。

## 5.2. 处理器控制的启动序列

启动只读存储器包含处理器复位后执行的第一条指令。两个处理器同时从相同位置进入启动只读存储器，但启动序列主要由核心0执行。

核心1在启动序列的早期阶段即被重定向至低功耗状态，等待核心0上的用户软件启动。若核心1未被使用，则保持在该低功耗状态。

### 源代码参考

本节描述的序列由启动只读存储器源代码仓库中的`arm8_bootrom_rt0.S`和`varm_boot_path.c`两个源文件实现。RISC-V内核则从`riscv_bootrom_rt0.S`开始，但与Arm共享启动路径的实现。

### 5.2.1. 启动结果

bootrom基于以下系统状态决定启动结果：

- 芯片选择信号0上的附加QSPI存储设备内容（如有）
- POWMAN寄存器BOOT0至BOOT3的内容
- 看门狗寄存器SCRATCH4至SCRATCH7的内容
- OTP内容，尤其是CRIT1、BOOT\_FLAGS0和BOOT\_FLAGS1
- QSPI CSn引脚被外部拉低（以选择BOOTSEL）
- QSPI SD1引脚被外部拉高（以在BOOTSEL模式下选择UART启动）

基于此，启动序列的结果可能为：

- 通过最近一次重启前SCRATCH或BOOT寄存器中指定的vector调用代码，例如进入从低功耗状态上电后保留在RAM中的代码。
- 从外部闪存运行映像。
  - 这可以通过两种方式之一实现：映像要么直接从外部闪存原地运行，要么在启动过程中被加载到RAM中。
  - RP2354上的封装内闪存在启动时视为外部存储。它是一个独立的硅片，RP2350芯片并不默认信任它。
- 运行预先加载至SRAM中的映像（不同于向量表情况）。
- 从OTP加载映像至SRAM并运行。
- 进入USB引导程序。
- 进入UART引导程序。
- 执行通过reboot() API请求的一次性操作，例如闪存更新启动。此操作可能由

用户，或UART或UF2引导程序发起请求。

- 因缺少合适映像且通过OTP禁用UART和USB引导程序，拒绝启动。

本节不区分不同类型的闪存启动（闪存映像启动、闪存分区启动及闪存分区表映像启动）。同样，本节不区分这些类型与在启动时加载至RAM的封装二进制文件，因其本质上是具有特殊加载映射的闪存二进制文件。本节仅描述启动只读存储器进行启动介质选择时的决策流程。

### 5.2.2. 启动序列

本节列举了Arm处理器控制的启动序列步骤。Arm与RISC-V存在一些细微差异，详见第5.2.2.2节。

[表451中所示的有效镜像](#)，指包含有效块循环且其中之一为有效镜像定义的镜像。在安全的RP2350上，该镜像必须（正确）签名，且必须满足所有其他安全要求，如最低回滚版本。

表451中操作列的阴影单元指示启动结果，详见第5.2.1节。其他单元为继续执行的过渡状态。两个内核均从 [Entry](#) 开始此序列。

表451中的主要顺序步骤如下：

- [Entry](#)
- [核心1等待](#)
- [启动路径开始](#)
- [等待救援](#)
- [生成启动随机数](#)
- [检查POWMAN向量](#)
- [检查看门狗向量](#)
- [准备启动镜像](#)
- [尝试RAM镜像启动](#)
- [检查BOOTSEL](#)
- [尝试OTP启动](#)
- [尝试Flash启动](#)
- [准备BOOTSEL](#)
- [进入USB启动](#)
- [进入UART启动](#)
- [启动失败](#)

关于表451的伪代码形式摘要，详见第5.2.2.1节。

表451。处理器控制的启动序列

| 条件（如果...）    | 操作（则...）                     |
|--------------|------------------------------|
| <b>步骤：入口</b> |                              |
| 始终           | 检查 CPUID 或 MHARTID 中的核心编号。   |
| 在核心0上运行      | 清除启动RAM（核心1区域及始终区域除外）。       |
|              | <a href="#">转至启动路径开始。</a>    |
| 在核心1上运行      | <a href="#">前往Core 1 等待。</a> |

| 条件 (如果...)                      | 操作 (则...)                                                                                                             |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>步骤: Core 1 等待</b>            |                                                                                                                       |
| 始终                              | 等待核心0将RCP salt寄存器标记为有效。                                                                                               |
|                                 | 等待核心0通过安全SIO FIFO提供入口点，使用第5.3节描述的协议。                                                                                  |
|                                 | 结果：设置安全主 <code>sp</code> 和 <code>VTOR</code> ，然后跳转至提供的入口点。                                                            |
| <b>步骤: 启动路径开始</b>               |                                                                                                                       |
| 始终                              | 检查救援标志，CHIP_RESET.RESCUE_FLAG                                                                                         |
| 救援标志已设置                         | 前往等待救援。                                                                                                               |
| 救援标志已清除                         | 前往生成启动随机数。                                                                                                            |
| <b>步骤: 等待救援</b>                 |                                                                                                                       |
| 始终                              | 清除救援标志以确认请求。                                                                                                          |
|                                 | 结果：在原地暂停。调试器连接以向处理器提供进一步指令。                                                                                           |
| <b>步骤: 生成启动随机数</b>              |                                                                                                                       |
| 始终                              | 将TRNG ROSC采样输入SHA-256以生成每次启动的256位随机数。                                                                                 |
|                                 | 在启动RAM中存储128位以供get_sys_info()检索，余数分配至RCP salt寄存器。                                                                     |
|                                 | 前往检查 POWMAN 向量。                                                                                                       |
| <b>步骤: 检查 POWMAN 向量</b>         |                                                                                                                       |
| 始终                              | 读取 BOOT0 至 BOOT3 以确定请求的启动类型。                                                                                          |
| 启动类型奇偶校验有效                      | 清除 BOOT0，以便在后续启动时忽略该值。                                                                                                |
| 已设置 <code>BOOTDIS</code> 标志     | 前往查看门狗向量。                                                                                                             |
| 启动类型为 <code>VECTOR</code>       | 结果：设置安全主 <code>sp</code> ，然后调用提供的入口点。                                                                                 |
| (从 <code>VECTOR</code> 返回)      | 前往查看门狗向量。                                                                                                             |
| 其他或无效的启动类型                      | 前往查看门狗向量。                                                                                                             |
| <b>步骤: 查看门狗向量</b>               |                                                                                                                       |
| 始终                              | 读取看门狗 SCRATCH4 至 SCRATCH7 以确定请求的启动类型。                                                                                 |
| 启动类型奇偶校验有效                      | 清除 SCRATCH4，以便在后续启动时忽略该值。                                                                                             |
| 启动类型为 <code>BOOTSEL</code>      | 备注：相当于通过拉低 QSPI <code>C<sub>n</sub></code> 线来选择 <code>BOOTSEL</code> 。                                                |
| 已设置 <code>BOOTDIS</code> 标志     | 前往准备镜像启动（因此当 <code>OTP BOOTDIS.NOW</code> 或 <code>POWMAN BOOTDIS.NOW</code> 标志被设置时， <code>BOOTSEL</code> 为唯一允许的启动类型）。 |
| 启动类型为 <code>VECTOR</code>       | 结果：设置安全主 <code>sp</code> ，然后调用提供的入口点。                                                                                 |
| (从 <code>VECTOR</code> 返回)      | 前往准备镜像启动。                                                                                                             |
| 启动类型为 <code>RAM_IMAGE</code>    | 请注意：此操作请求扫描已预装映像的RAM区域。                                                                                               |
| 启动类型为 <code>FLASH_UPDATE</code> | 请注意：修改部分闪存启动行为，详见第5.1.16节。                                                                                            |
| 始终                              | 前往准备镜像启动。                                                                                                             |
| <b>步骤: 准备映像启动</b>               |                                                                                                                       |

| 条件（如果...）                                                     | 操作（则...）                                                                                                   |
|---------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| 始终                                                            | 清除 <b>BOOTDIS</b> 标志（OTP BOOTDIS.NOW 和 POWMAN BOOTDIS.NOW）。                                                |
|                                                               | 为SRAM0和SRAM1供电（XIP RAM域已通电）。                                                                               |
|                                                               | 复位所有PADS和IO寄存器，并解除QSPI PAD隔离。                                                                              |
|                                                               | 释放USB复位并清除USB RAM上部3 KB（用于搜索工作区）。                                                                          |
|                                                               | 进入尝试RAM映像启动。                                                                                               |
| <b>步骤：尝试RAM映像启动</b>                                           |                                                                                                            |
| 看门狗类型不为<br><b>RAM_IMAGE</b>                                   | 进入检查BOOTSEL。                                                                                               |
| <b>BOOT_FLAGS0.DISABLE_SR_A</b><br><b>M_WINDOW_BOOT</b> 标志被设置 | 进入检查BOOTSEL。                                                                                               |
| 否则                                                            | 扫描指定的RAM地址范围以查找有效映像（起始地址存于SCRATCH2，长度存于SCRATCH3）。此操作用于启动通过UF2下载的RAM映像。                                     |
| RAM 镜像有效                                                      | 结果：按照镜像定义中规定的方式输入 RAM 镜像。                                                                                  |
| 无有效镜像                                                         | 前往 <b>准备进入 BOOTSEL</b> （跳过 Flash 和 OTP 启动）。                                                                |
| <b>步骤：检查 BOOTSEL</b>                                          |                                                                                                            |
| 始终                                                            | 检查 BOOTSEL 请求：QSPI CSn 为低电平（BOOTSEL 按钮），看门狗类型为 <b>BOOTSEL</b> ，或检测到 RUN 引脚双击（由 BOOT_FLAGS1.DOUBLE_TAP 启用）。 |
| 请求 BOOTSEL                                                    | 前往 <b>准备进入 BOOTSEL</b> （跳过 Flash 和 OTP 启动）。                                                                |
| 否则                                                            | 前往尝试 OTP 启动。                                                                                               |
| <b>步骤：尝试 OTP 启动</b>                                           |                                                                                                            |
| 始终                                                            | 检查 <b>BOOT_FLAGS0.DISABLE OTP_BOOT</b> 和 <b>BOOT_FLAGS0.ENABLE OTP_BOOT</b> （禁用项优先）。                       |
| OTP 启动已禁用                                                     | 前往尝试 Flash 启动。                                                                                             |
| OTP 启动已启用                                                     | 从 OTPBOOT_SRC（OTP 区）加载数据到 OTPBOOT_DST0/OTPBOOT_DST1（SRAM 区），长度由 OTPBOOT_LEN 指定。                            |
|                                                               | 在 SRAM 中就地检查映像的有效性。                                                                                        |
| 映像有效                                                          | 结果：按照镜像定义中规定的方式输入 RAM 镜像。                                                                                  |
| 无有效镜像                                                         | 前往尝试 Flash 启动。                                                                                             |
| <b>步骤：尝试闪存启动</b>                                              |                                                                                                            |
| 由<br><b>BOOT_FLAGS0 禁用闪存启动</b>                                | 前往准备进入 BOOTSEL。                                                                                            |
| 始终                                                            | 向片选0发出XIP退出序列。                                                                                             |
| <b>FLASH_DEVINFO</b> 包含片选1的GPIO和尺寸信息                          | 向片选1发出XIP退出序列。                                                                                             |
| 始终                                                            | 扫描闪存以查找有效映像（可能位于分区内），使用指令（EBh, BBh、0Bh、03h）及SCK分频（3至24）                                                    |

| 条件（如果...）              | 操作（则...）                                                                                                                                                                                                                                                                                                                                       |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 找到有效映像                 | <p>结果：按照映像定义规定的方式进入闪存映像。这可能包括将部分闪存内容加载至RAM。</p> <p>将当前闪存读取模式保存为XIP配置函数，置于引导RAM底部，后续可调用以恢复当前模式（例如，串行编程后）。</p>                                                                                                                                                                                                                                   |
| 无有效镜像                  | 前往准备进入 BOOTSEL。                                                                                                                                                                                                                                                                                                                                |
| <b>步骤：准备进入 BOOTSEL</b> |                                                                                                                                                                                                                                                                                                                                                |
| 始终                     | <p>在将内存及外设交付非安全状态之前，将SRAM0至SRAM9、XIP缓存和USB RAM全部清零。</p> <p>启用 XOSC 并根据 BOOTSEL_XOSC_CFG 和 BOOTSEL_PLL_CFG 配置 PLL 至 48 MHz（默认预期为 12 MHz 晶体）。</p> <p>检查带默认下拉电阻的 QSPI SD1 引脚以选择 UART/USB 启动。</p> <p>扫描闪存中的分区表（始终使用 03h 串行读取命令且 SCK 除数为 6）。USB 启动加载程序可能根据分区及其内容，将 UF2 下载到不同的闪存地址。</p> <p>如软锁定的 OTP 状态比其 S 状态更加严格，则将所有 OTP 软锁定推进至 OTP 中的 BL 状态。</p> |
| QSPI SD1 被拉低           | 进入USB 启动。                                                                                                                                                                                                                                                                                                                                      |
| QSPI SD1 被拉高           | 进入UART 启动。                                                                                                                                                                                                                                                                                                                                     |
| <b>步骤：进入 USB 启动</b>    |                                                                                                                                                                                                                                                                                                                                                |
| 始终                     | 检查 BOOT_FLAGS0.DISABLE_BOOTSEL_USB_PICOBOOT_IFC 与 BOOT_FLAGS0.DISABLE_BOOTSEL_USB_MSD_IFC 以确定允许的 USB 接口。                                                                                                                                                                                                                                       |
| 两个USB接口已禁用             | 前往启动失败。                                                                                                                                                                                                                                                                                                                                        |
| 否则                     | 结果：进入USB引导程序。若下载了UF2镜像，引导程序将重启，并在看门狗暂存寄存器中标记一个FLASH_UPDATE（如适用），引导路径将从 Entry重新启动。有效镜像可引导；无效镜像通常会返回USB引导程序。                                                                                                                                                                                                                                     |
| <b>步骤：进入UART引导</b>     |                                                                                                                                                                                                                                                                                                                                                |
| 始终                     | 检查BOOT_FLAGS0.DISABLE_BOOTSEL_UART_BOOT以确认是否允许UART引导。                                                                                                                                                                                                                                                                                          |
| UART引导已禁用              | 前往启动失败。                                                                                                                                                                                                                                                                                                                                        |
| 否则                     | 结果：进入UART引导程序。下载镜像后，引导程序将重启，启动类型为 RAM_IMAGE，引导路径将从 Entry重新开始。有效镜像可引导；无效镜像通常会返回UART引导程序。                                                                                                                                                                                                                                                        |
| <b>步骤：启动失败</b>         |                                                                                                                                                                                                                                                                                                                                                |
| 始终                     | 结果：不采取进一步操作。未发现有效启动镜像，且所选BOOTSEL接口已被禁用。连接调试器以向处理器提供进一步指令。查看启动RAM中的启动原因，以诊断启动失败的原因。                                                                                                                                                                                                                                                             |

### 💡 提示

引导只读存储器（bootrom）内部将 BOOTSEL 模式称为 NSBOOT，原因在于 USB 和 UART 引导加载程序于 Arm 的非安全状态下运行。本章有时亦用 NSBOOT 表述 BOOTSEL。

#### 5.2.2.1. 引导序列伪代码

下列伪代码概述了表 451。

```

if (powman_vector_valid && powman_reboot_mode_is_pcsp) { //
 若 (correct_arch), 本调用可能返回并继续引导流程 powman_
 vector_pc(); else hang();
}
if (watchdog_vector_valid) {
 // 记录 RAM_IMAGE、FLASH_UPDATE、BOOTSEL 重启类型
 check_special_reboot_mode();
 if (watchdog_reboot_mode_is_pcsp) {
 //若 (correct_arch), 本调用可能返回并继续引导流程 watch
 dog_vector_pc(); else hang();
 }
}

// RAM 镜像窗口由 watchdog_scratch 指定, 例如 UF2 RAM // 下载后: 执行 RAM 镜像, 或
// 退至 UART/USB 引导。
if (watchdog_reboot_mode_is_ram_image && !ram_boot_disabled_in_otp) { //
 //仅当不存在有效的RAM映像可供进入时, 才会返回。
 //无法从RAM映像返回。
 try_boot_ram_image(ram_image_window);
} else {
 //否则尝试OTP和闪存启动 (除非有请求跳过)
 skip_flash_and_otp_boot =
 bootsel_button_pressed() ||
 watchdog_reboot_mode_is_bootsel ||
 (double_tap_enabled_in_otp() && double_run_reset_detected());

 if (!skip_flash_and_otp_boot) {
 if (otp_boot_enabled_in_otp && !otp_boot_disabled_in_otp) {
 //仅当不存在有效的OTP映像可供进入时, 才会返回。
 //无法从OTP映像返回。
 try_otp_boot();
 }
 if (!flash_boot_disabled_in_otp) {
 //仅当不存在有效的闪存映像可供进入时, 才会返回。
 //无法从闪存映像返回。
 try_flash_boot();
 }
 }
}
//未找到镜像, 因此切换到其中一个引导加载程序
if (sd1_high_select_uart) {
 //除非重新启动, 否则不返回
 if (nsboot_uart_disabled) hang(); else nsboot(uart);
} else {
 //除非重新启动, 否则不返回
 if (nsboot_usb_disabled) hang(); else nsboot(usb);
}

```

### 5.2.2.2. Arm与RISC-V的差异

表451中列出的引导序列在RISC-V上存在以下差异：

- 不支持安全启动（无论镜像来源为何）。
- 不支持防回滚检查，该项仅针对安全启动适用。
- 诸如使用RCP验证布尔值的额外安全检查已被禁用。
- UART和USB引导加载程序继续在机器模式下运行，未从Arm安全态切换至非安全态，意味着这些引导阶段间不存在硬件强制的安全边界。
- 成功闪存引导时写入引导RAM的XIP设置函数采用的是RISC-V指令，而非Arm指令。

### 5.2.3. POWMAN 启动向量

POWMAN 包含类似于看门狗临时寄存器的 scratch 寄存器，这些寄存器在切换核心电源域断电后仍然保持，且在大多数系统复位时亦保持不变。这些寄存器允许用户安装自定义引导处理程序，并在非 POR/BOR 复位时脱离主启动序列的控制流程。其识别写入 BOOT0 至 BOOT3 的以下数值：

- **BOOT0**: 魔数 `0xb007c0d3`
- **BOOT1**: 入口点与魔数异或-`0xb007c0d3` (`0x4ff83f2d`)
- **BOOT2**: 栈指针
- **BOOT3**: 入口点

利用此方法，可跳转至在低功耗状态下保留于 RAM 中预加载的代码。

若任一魔数不匹配，则不会执行 POWMAN 向量启动。若数值匹配，引导只读存储器将在进入向量前清零 BOOT0，以防该行为在后续重启时持续。

POWMAN 启动向量允许返回。从POWMAN向量引导返回后，引导序列依旧正常继续，仿佛向量引导未曾执行。RISC-V 架构下，向量不要求保护全局指针（`gp`）寄存器。可利用此特性执行启动路径的其他必要配置，例如向可能已断电的外部 QSPI 设备发送上电命令（如通过B9h断电命令）。

入口点（`pc`）在Arm架构上必须设置最低有效位（即Thumb位），而在RISC-V架构上必须清除该位。若未满足此条件，bootrom将假定您将RISC-V函数指针传递给Arm处理器（或反之），并使核心挂起，避免继续执行，因错误架构代码的执行将产生严重的不确定性后果。

链接器应自动为函数指针重定位正确设置Thumb位，但若传递硬编码值，如SRAM基址，需特别注意：在Arm架构中应传递为 `0x20000001`（Thumb位已设置），而为 `0x20000000` 基于RISC-V架构（无Thumb位，半字对齐）。

### 5.2.4. 看门狗启动向量

看门狗启动允许用户安装其自定义启动处理程序，并在非上电复位（POR）或掉电复位（BOR）情况下，将控制权从主启动序列转移。它识别写入看门狗上部临时寄存器的以下数值：

- **SCRATCH4**: 魔数 `0xb007c0d3`
- **SCRATCH5**: 入口点与魔数异或-`0xb007c0d3` (`0x4ff83f2d`)
- **SCRATCH6**: 堆栈指针
- **SCRATCH7**: 入口点

若任一魔数不匹配，则不执行看门狗启动。若数值匹配，Bootrom

在转移控制权前将SCRATCH4清零，以确保该行为不会在后续重启中持续。

看门狗启动亦可用来选择Bootrom的特殊一次性启动模式，该模式详见5.2.4.1节。术语 **one-shot** 意味着此类启动模式仅影响下一次启动（不影响后续启动），因Bootrom在每次启动时均会清除SCRATCH4。这些启动类型通过设置特殊入口点（`pc`）值为 `0xb007c0d3` 进行编码，该值在其他情况下并非有效入口地址，然后在堆栈指针（`sp`）值中设置启动类型。第5.2.4.1节列出了支持的值。

看门狗启动向量允许返回。返回时启动路径将正常继续：可借此执行启动路径所需的任何额外配置，例如向外部QSPI设备发出附加命令。在RISC-V架构中，向量允许使用其自身的全局指针（`gp`）值，因为启动只读存储器仅在USB启动期间使用 `gp`，并安装其自身的值。

除特别启动类型入口点（`0xb007c0d3`）外，向量入口点 `pc` 在Arm上必须设置最低有效位（Thumb位），而在RISC-V上必须清除该位。若不满足此条件，启动只读存储器将假定您向Arm处理器传递了RISC-V函数指针（或反之），并挂起内核而非继续执行。

#### 5.2.4.1. 特殊看门狗启动类型

魔术入口点 `0xb007c0d3` 指示一种特殊的一次性启动类型，其标识由栈指针值确定：

##### BOOTSEL

由 `sp = 2` 选择。进入BOOTSEL模式启动。启动方式为UART或USB，具体取决于QSPI SD1是否被拉高（默认下拉电阻选择USB启动）。详情请参见第5.2.8节。

##### RAM\_IMAGE

由 `sp = 3` 选择。启动存储于SRAM或XIP SRAM中的映像。BOOTSEL模式通过此方式请求执行在重启前加载到RAM的映像。详情请参见第5.2.5节。

##### FLASH\_UPDATE

由 `sp = 4` 选择。BOOTSEL在闪存下载后重启时选择此模式。更改部分闪存启动行为，例如允许旧版本启动优先于新版本。详情请参见第5.1.16节。

一次性启动类型的参数如下传入：

- SCRATCH2：参数0
- SCRATCH3：参数1

这些直接对应于传递给reboot() API的 `p0` 和 `p1` 启动参数。例如，在 RAM\_IMAGE 启动时，这指定用于搜索有效IMAGE\_DEF 的 RAM 区域的起始地址和大小。详情请参见第5.4.8.24节的API列表。当未执行所列任一启动类型时，SCRATCH2 和 SCRATCH3 保留为用户可自由使用的值，启动只读存储器不会修改或解释其内容。

#### 5.2.5. RAM 镜像启动

启动只读存储器通过看门狗寄存器中的值指示启动至SRAM或XIP SRAM中的图像。两个参数指示用于搜索包含有效（必要时经正确签名）IMAGE\_DEF 块环的区域的起始地址和大小。这些作为参数0/1传入，在看门狗擦写区2/3中。

如果待启动的映像包含于XIP SRAM中，bootrom必须在启动前将XIP SRAM固定到位。因此，若您使用XIP SRAM存放二进制文件，必须在 LOAD\_MAP 条目中添加特殊项（详见第5.9.3.2节）。

### 5.2.6. OTP 启动

若启用OTP启动，则优先执行OTP中的代码，而非闪存中的代码。请注意，OTP代码可自由“链入”存储于闪存中的可执行程序。

OTP中的代码被复制到SRAM指定位置，随后执行流程类似于RAM映像启动。

在复制自OTP的SRAM中搜索有效的（若必要且已正确签名）[IMAGE\\_DEF](#)，找到则启动；否则，OTP启动将退回执行闪存启动（若已启用）。

一次性可编程（OTP）启动代码例如可用于执行隐藏的解密代码，以在启动时解码闪存映像。一旦完成，OTP启动代码甚至可以在安全代码中将自身隐藏（存于OTP中）。

### 5.2.7. 闪存启动

只读存储器启动程序会扫描闪存最多16次，直到找到有效的[IMAGE\\_DEF](#)或[PARTITION\\_TABLE](#)。此时，闪存设置被视为有效，如在这些设置下找到有效的可启动[IMAGE\\_DEF](#)，则闪存启动程序将继续执行。它使用以下闪存读取指令与SCK除数组合进行16次尝试：

表452。只读存储器按尝试顺序支持的QSPI/读取模式。

| 模式       | 时钟除数 |
|----------|------|
| EBh 四线模式 | 3    |
| BBh 双线模式 | 3    |
| 0Bh 串行模式 | 3    |
| 03h 串行模式 | 3    |
| EBh 四线模式 | 6    |
| BBh 双线模式 | 6    |
| 0Bh 串行模式 | 6    |
| 03h 串行模式 | 6    |
| EBh 四线模式 | 12   |
| BBh 双线模式 | 12   |
| 0Bh 串行模式 | 12   |
| 03h 串行模式 | 12   |
| EBh 四线模式 | 24   |
| BBh 双线模式 | 24   |
| 0Bh 串行模式 | 24   |
| 03h 串行模式 | 24   |

QSPI无法提供可靠方法检测设备是否已连接。不过，这对启动过程来说并非重大问题：要么存在具有有效且可启动内容的设备，要么不存在此类内容（可能由于设备未连接、设备内容无效，或当前QSPI模式下通信失败）。

当无设备（或无可识别内容）时，bootrom 会依次尝试表 452 中的全部 16 种模式，最终放弃。初始搜索区域限制为 4 kB，以最大限度减少扫描闪存所耗费的时间，随后再尝试 USB 或 UART 启动。该 4 kB 限制同样适用于闪存分区内的搜索，使得 bootrom 能够通过在分区起始处擦除单个 4 kB 扇区，可靠切断所含映像的块循环，例如用于版本降级。

bootrom 定位闪存映像的三种主要方式为：

### 闪存映像启动

闪存映像可直接写入闪存存储地址 `0x0`, bootrom 将从该地址查找映像。此方式最类似于 RP2040 上的闪存启动（主要差别是映像前 256 字节内不再包含 `boot2`, 并且新增要求映像前 4 kB 内必须存在有效的映像定义）。

### 闪存分区启动

闪存映像可写入分区表中的某个分区。分区表由存储于闪存起始处的 `PARTITION_TABLE` 块描述。启动只读存储器定位分区表并扫描其分区以查找可启动映像。

### 镜像中分区表启动

包含 `IMAGE_DEF` 与 `PARTITION_TABLE` 块的闪存映像，在单个循环块中写入闪存起始位置。

启动只读存储器加载嵌入的分区表，并以与闪存映像启动相同的方式加载映像情况。

请重新查阅相关启动只读存储器概念章节，以全面理解这三种闪存启动方式。

本节关注启动只读存储器是否能发现有效可启动映像。在三种情形中，映像须具备有效的 `IMAGE_DEF`，且满足所有相关安全要求，例如正确签名及回滚版本不低于 OTP 存储版本。

只读存储器启动程序以在闪存编程期间发现的任何可用 QSPI 模式进入闪存镜像。任何后续设置（例如无前缀连续读取模式）由闪存镜像本身执行。此设置代码称为 **XIP 设置函数**，通常在执行前复制到 RAM 中，以避免在重新配置 XIP 接口时从闪存执行。

#### 💡 提示

SDK 中的 `PICO_EMBED_XIP_SETUP=1` 标志启用在 RP2350 构建中包含并执行 XIP 设置函数。在此情况下，该函数在早期启动阶段于核心 0 堆栈上执行，因此无需分配额外的静态内存。后续调用情况不同，因为启动后堆栈通常不可执行。

您应将 XIP 设置函数保存在启动 RAM 的前 256 字节内，以便在因串行闪存编程操作必须退出 XIP 模式后重新初始化 XIP 模式时能够轻松定位。bootrom 在进入闪存映像之前，会将默认的 XIP 设置函数写入该地址，以恢复 bootrom 在闪存编程期间检测到的模式。

#### ⚠ 注意

启动 RAM 不可执行，因此您无法直接从启动 RAM 执行 XIP 设置函数。必须先将其复制到 SRAM 中方可执行。

XIP 设置函数应完全位置无关，且大小不得超过 256 字节。如无法满足上述要求，应安装一个存根函数，在 RAM 其他位置调用该 XIP 设置函数。

### 5.2.8. BOOTSEL (USB/UART) 启动

bootrom 在复位后不久采样 QSPI `CSn` 的状态。bootrom 根据采样结果决定是否进入 BOOTSEL 模式，该模式包括 USB 和 UART 启动加载程序。

bootrom 将芯片选择信号初始化为以下状态：

- 输出禁止
- 被拉高（注意 `CSn` 为低电平有效信号，故此操作取消对外部 QSPI 设备的选择（若存在））

若芯片选择保持高电平，bootrom 则继续其正常的非 BOOTSEL 流程。默认情况下，对于空白设备，这意味着将芯片选择信号拉低并尝试从外部闪存或 PSRAM 设备启动。

如果芯片选择信号由外部拉低，bootrom 将进入 BOOTSEL 模式。您必须以

足够低的阻抗拉低芯片选择信号，以克服内部的上拉电阻。一个接地的4.7 kΩ电阻是一个合适的中间值，能够可靠地产生低电平输入逻辑，但不会影响RP2350驱动芯片选择信号时的输出电平。

QSPI SD1 线由RP2350初始拉低，该线用于选择进入的引导加载程序：

- SD1 保持拉低：进入USB引导加载程序
- SD1 被拉高：进入UART引导加载程序

USB引导是一种操作简便的方法，可通过如Linux PC等高级主机对RP2350进行编程，同时还直接支持更高级的选项，如OTP编程。详见第5.5节拖放式大容量存储接口，或第5.6节的PICOBOOT供应商接口。

UART引导是用于从另一微控制器启动无闪存RP2350的最小接口。UART引导使用QSPI SD2作为UART TX，使用QSPI SD3作为UART RX，波特率固定为1 Mbaud。有关UART引导的更多详细信息，请参见第5.8节。

### 5.2.8.1. BOOTSEL时钟要求

BOOTSEL模式要求在 XIN 和 XOUT 引脚之间连接晶体，或由外部振荡器输出的时钟信号输入至 XIN 引脚。有关这两个XOSC 引脚的电气规格，请参见表1439。

Bootrom假定XOSC的默认频率为12 MHz，并配置USB PLL以从XOSC参考中派生固定的48 MHz频率。对于USB，此频率必须极其精确。如果您使用非12 MHz晶体且计划启用USB引导，请在OTP中编程BOOTSEL\_PLL\_Cfg 和 BOOTSEL\_XOSC\_Cfg，并设置BOOT\_FLAGS0.ENABLE\_BOOTSEL\_NON\_DEFAULT\_PLL\_XOSC\_Cfg。有关针对您的晶体计算正确PLL参数的详细说明，请参见第8.6.3节。

UART启动使用与USB启动相同的PLL配置。然而，在默认PLL配置下，晶振频率的允许范围更广。详见第5.8.1节。

### 5.2.9. 启动配置 (OTP)

存储在OTP中的用户配置详见第13.10节，自CRIT1起。

启动只读存储器(bootrom)的主要控制位存于BOOT\_FLAGS0和BOOT\_FLAGS1，二者均位于OTP的第1页，该页在空白设备上的默认权限如下：

- 安全区(S) 可读写
- 引导加载程序(BL) 可读写
- 非安全区(NS) 只读

启动密钥哈希存储于OTP第2页，从BOOTKEY0\_0开始。该页最多可存放四个启动密钥哈希。关于密钥安装的示例，详见第5.10.1节。

## 5.3. 在处理器核1上启动代码

如第5.2节所述，复位后，处理器核心1启动时保持休眠，直至由核心0通过SIO FIFO唤醒。

如果您正在使用SDK，您可以使用`multicore_launch_core1()`函数在处理器核心1上启动代码。  
本节描述了自行在处理器核心1上启动代码的具体步骤。

在处理器核心1上启动运行的步骤涉及两个核心通过状态机协调同步执行，该状态机通过处理器间FIFO传递消息实现控制。此状态机设计稳健，可处理刚重置的处理器核心1，无论其启动代码执行到了哪一阶段，包括进入睡眠状态。因此，该步骤可在处理器核心1完成重置后（无论是系统重置，

还是仅显式重置处理器核心 1) 任意时刻执行。以下 C 代码说明了该步骤：

```
// 从核心 0 到核心 1 顺序发送的 FIFO 值
//
// vector_table 为 VTOR 寄存器的值
// sp 是初始堆栈指针 (SP)
// entry 是初始程序计数器 (PC) (别忘了设置 thumb 位!)
const uint32_t cmd_sequence[] =
 {0, 0, 1, (uintptr_t) vector_table, (uintptr_t) sp, (uintptr_t) entry};

uint seq = 0;
do {
 uint cmd = cmd_sequence[seq];
 // 在发送0之前，始终清空读取FIFO (来自核心1)
 if (!cmd) {
 // 丢弃读取FIFO中的数据，直到为空
 multicore_fifo_drain();
 // 执行SEV指令，因为核心1可能正在等待FIFO空间
 __sev();
 }
 // 向写入FIFO写入32位值
 multicore_fifo_push_blocking(cmd);
 // 一旦可用，从读取FIFO读取32位值
 uint32_t response = multicore_fifo_pop_blocking();
 // 在获得正确响应 (回显值) 时转到下一状态，否则重新开始
 seq = cmd == response ? seq + 1 : 0;
} while (seq < count_of(cmd_sequence));
```

## 5.4. Bootrom API

尽管部分只读存储器空间专门用于实现引导序列及USB/UART引导接口，bootrom中亦包含提供有用RP2350功能的公共函数，这些函数可能对设备上运行的任何代码或运行时环境具有参考价值。

分类列表详见第5.4.6节。

完整字母顺序列表详见第5.4.7节。

### 5.4.1. 定位 API 函数

API函数通常通过SDK中的封装函数提供给用户，然而，为了适配其他运行时环境或满足用户直接定位函数位置的需求，亦提供了较低级别的方法来查找这些函数（因其地址可能随bootrom版本变动）。

表453展示了在采用Arm架构时，用于定位这些函数的bootrom中特定字的固定内存布局。表454展示了在采用RISC-V架构时可用的附加条目。

表453。用于Arm代码的bootrom内容  
固定（众所周知）  
地址

| 地址         | 内容    | 描述               |
|------------|-------|------------------|
| 0x00000000 | 32位指针 | 初始启动堆栈指针         |
| 0x00000004 | 32位指针 | 指向启动复位处理程序函数的指针  |
| 0x00000008 | 32位指针 | 指向启动NMI处理程序函数的指针 |
| 0x0000000c | 32位指针 | 指向启动硬故障处理程序函数的指针 |

| 地址         | 内容             | 描述                                     |
|------------|----------------|----------------------------------------|
| 0x00000010 | 'M', 'u', 0x02 | 魔数                                     |
| 0x00000013 | 字节             | Bootrom版本                              |
| 0x00000014 | 16位指针          | 指向只读存储器入口表的指针 (BOOTROM_ROMTABLE_START) |
| 0x00000016 | 16位指针          | 指向辅助函数的指针 (rom_table_lookup_val())     |
| 0x00000018 | 16位指针          | 指向辅助函数的指针 (rom_table_lookup_entry())   |

表454。RISC-V代码的只读存储器内容位于固定（广为人知的）地址

| 地址         | 内容    | 描述                                     |
|------------|-------|----------------------------------------|
| 0x00007df6 | 16位指针 | 指向只读存储器入口表的指针 (BOOTROM_ROMTABLE_START) |
| 0x00007df8 | 16位指针 | 指向辅助函数的指针 (rom_table_lookup_val())     |
| 0x00007dfa | 16位指针 | 指向辅助函数的指针 (rom_table_lookup_entry())   |
| 0x00007dfc | 32位指令 | RISC-V入口点                              |

假设从地址 0x00000010 开始的三个字节为 ('M'、 'u'、 0x02)，则其他固定位置字段可视为有效，并可用于查找只读存储器功能。

偏移量 0x00000013 处的版本字节仅供参考，不应用于推断任何函数的精确位置。该字节对 A2 硅片的值为 2。

以下 SDK 代码展示了 SDK 如何查找只读存储器中的函数：

```
static __force_inline void *rom_func_lookup_inline(uint32_t code) {
#ifndef __riscv
 // 在 RISC-V 架构上，代码 (jmp) 实际上嵌入于表中
 rom_table_lookup_fn rom_table_lookup =
 (rom_table_lookup_fn) (uintptr_t)*(uint16_t*)(BOOTROM_TABLE_LOOKUP_ENTRY_OFFSET
 + rom_offset_adjust);
 return rom_table_lookup(code, RT_FLAG_FUNC_RISCV);
#else
 // 在 Arm 架构中，函数指针存储于表中，因此通过 lookup() 函数而非 lookup_entry() 函数进行
 // 解引用
 rom_table_lookup_fn rom_table_lookup =
 (rom_table_lookup_fn) (uintptr_t)*(uint16_t*)(BOOTROM_TABLE_LOOKUP_OFFSET);
 if (pico_processor_state_is_nonsecure()) {
 return rom_table_lookup(code, RT_FLAG_FUNC_ARM_NONSEC);
 } else {
 return rom_table_lookup(code, RT_FLAG_FUNC_ARM_SEC);
 }
#endif
}
```

除了 API 函数外，亦可查询部分数据值。以下代码示例：

```
void *rom_data_lookup(uint32_t code) {
 rom_table_lookup_fn rom_table_lookup =
 (rom_table_lookup_fn) (uintptr_t)*(uint16_t*)(BOOTROM_TABLE_LOOKUP_OFFSET);
 return rom_table_lookup(code, RT_FLAG_DATA);
}
```

参数 code 对应下表中的 CODE 值，具体计算方法如下：

```
uint32_t rom_table_code(char c1, char c2) {
 return (c2 << 8) | c1;
}
```

这些代码也可在 [SDK](#) 中的 `bootrom.h` 文件中以 `#define` 形式获得。

### 5.4.2. API 函数可用性

某些函数并非所有架构或安全级别均支持。第5.4.6节的 API 列表采用以下术语标明各个 API 入口点的可用性：

#### Arm-S

该函数适用于安全 Arm 代码。除专门涉及 RISC-V 或非安全功能外，大多数函数对 [Arm-S](#) 均可用。

#### RISC-V

该函数适用于 RISC-V 代码。除专门涉及 Arm 安全状态外，大多数在 [Arm-S](#) 下可用的函数在 RISC-V 下也适用。

#### Arm-NS

该函数适用于非安全 Arm 代码。该函数在此情况下执行额外的权限和参数检查，以防止安全数据泄露或损坏。

每个单独的 [Arm-NS](#) API 函数必须由 Secure 代码显式启用后方可使用，方法是调用 `set_ns_api_permission()`。若被 Secure 代码禁用，禁用的非安全 API 将返回 `BOOTROM_ERROR_NOT_PERMITTED`。所有非安全 API 初始状态均为禁用。非安全代码调用仅限 Secure 的 Arm-S 函数时无权限控制，但此类调用在尝试访问仅限 Secure 的硬件时将导致崩溃。

通过 Secure Gateway (SG) 指令，[Arm-NS](#) 函数可提升调用权限，允许非安全代码对名义上仅限 Secure 的硬件执行有限操作，例如用于闪存编程的 QSPI 直通模式接口。

对于 [RISC-V](#) 函数而言，基于特权级别无单独入口点。假设在 PMP 中拥有 ROM 地址的执行权限，M 模式和 U 模式软件均可调用 bootrom API，但若 U 模式调用尝试访问仅限 M 模式的硬件，则会导致崩溃。

### 5.4.3. API 函数返回码

某些函数不支持返回任何错误，标记为 `void`。其余返回值为 0 (`BOOTROM_OK`) 或正值（如需返回数据）表示成功。这些 bootrom 错误代码与 SDK 使用的错误代码完全相同，故可互换使用。这解释了 SDK 错误代码编号中未被 bootrom 使用的编号间隙。

| 名称                                       | 值        | 描述                                                                                                                    |
|------------------------------------------|----------|-----------------------------------------------------------------------------------------------------------------------|
| 值                                        | $\geq 0$ | 函数成功执行，返回该值                                                                                                           |
| <code>BOOTROM_OK</code>                  | 0        | 函数执行成功                                                                                                                |
| <code>BOOTROM_ERROR_NOT_PERMITTED</code> | -4       | 该操作因安全约束被禁止                                                                                                           |
| <code>BOOTROM_ERROR_INVALID_ARG</code>   | -5       | 传递给函数的一个或多个参数超出支持范围；<br><code>BOOTROM_ERROR_INVALID_ADDRESS</code> 和 <code>OOTROM_ERROR_BAD_ALIGNMENT</code> 为更具体的错误。 |

| 名称                                     | 值   | 描述                                       |
|----------------------------------------|-----|------------------------------------------|
| BOOTROM_ERROR_INVALID_ADDRESS          | -10 | 地址参数越界，或被判定为调用者无权访问的地址。                  |
| BOOTROM_ERROR_BAD_ALIGNMENT            | -11 | 传入地址未正确对齐。                               |
| BOOTROM_ERROR_INVALID_STATE            | -12 | 过去发生或未发生某事，因而请求当前无法处理。                   |
| BOOTROM_ERROR_BUFFER_TOO_SMALL         | -13 | 用户分配的缓冲区过小，无法容纳函数的结果或工作状态。               |
| BOOTROM_ERROR_PRECONDITION_NOT_MET     | -14 | 调用失败，因为必须先调用另一个只读存储器函数。                  |
| BOOTROM_ERROR_MODIFIED_DATA            | -15 | 缓存的数据被判定与其复制的完整版本数据不一致。                  |
| BOOTROM_ERROR_INVALID_DATA             | -16 | 数据结构内容无效。                                |
| BOOTROM_ERROR_NOT_FOUND                | -17 | 尝试访问不存在的对象；或者，搜索失败。                      |
| BOOTROM_ERROR_UNSUPPORTED_MODIFICATION | -18 | 基于当前状态，无法进行修改。此情况可能发生于尝试清除一次性可编程（OTP）位时。 |
| BOOTROM_ERROR_LOCK_REQUIRED            | -19 | 所需的锁未被持有。详见第5.4.4节。                      |

#### 5.4.4. API 函数与独占访问权限

各种只读存储器函数需要访问系统的部分区域，这些区域：

- 不可由两个核心同时安全访问，或者
- 使用时会限制其他硬件的功能。

例如：

- 编程一次性可编程（OTP）：无法在访问映射的OTP数据区域的同时访问其串行编程接口。
- 使用SHA-256模块：一次只能进行一项SHA-256校验和操作。
- 使用QSPI直接模式接口对闪存进行编程会导致XIP访问返回总线错误。

启动只读存储器程序的职责范围不包括实现锁定策略，因锁定的类型和范围完全取决于应用程序对这些资源的具体使用方式。

然而，重要的是，例如非安全调用闪存编程API时，不应引起运行于闪存中的其他安全代码发生硬故障。用户软件必须具备某种方式与启动只读存储器程序API在状态变更时进行协调。启动只读存储器程序实现了机制但不负责制定策略以实现对启动只读存储器程序API调用的互斥访问。

启动只读存储器程序提供的解决方案是利用启动锁（启动RAM寄存器BOOTLOCK0至BOOTLOCK7）告知启动程序当前哪些资源归调用者所有，从而可安全使用。

要启用启动只读存储器程序API中的锁检测，请将启动锁 7 (`LOCK_ENABLE`) 设置为申明状态。启用后，使用某些硬件资源（如下所列）的 bootrom 函数将检查分配给该资源的启动锁状态，如果该锁不处于声明状态，则返回`BOOTROM_ERROR_LOCK_REQUIRED`。

在调用启用锁定的 bootrom 函数前，必须先声明相关锁。可能需要多次尝试

才能成功声明锁，尤其当该 API 被其他上下文并发访问时。声明锁时，请遵循与 SIO 自旋锁（第3.1.4节）相同的步骤。

以下启动锁已被分配：

- `0x0 : LOCK_SHA_256` - 拥有该锁时，bootrom API 允许使用 SHA-256 模块
- `0x1 :LOCK_FLASH_OP` - 拥有该锁时，bootrom API 允许在 QSPI 存储接口（第12.14.5节）进入直接模式，以执行低级闪存操作
- `0x2 :LOCK OTP` - 拥有该锁时，bootrom API 允许通过串行接口访问 OTP
- `0x7 :LOCK_ENABLE` - 若拥有，则启用只读存储器API资源所有权检查。此项默认关闭，因为只读存储器API旨在默认即可使用，无需额外设置。

### 5.4.5. SDK 对 API 的访问

只读存储器功能通过`pico_bootrom`库（详见`pico_bootrom`）在SDK中提供。

每个只读存储器函数均有`rom_`包装函数，该函数查找只读存储器函数地址并进行调用。

SDK通过`bootrom_acquire_lock_blocking(n)`和`bootrom_release_lock(n)`实现了简易的独占访问控制。默认启用时（定义了`PICO_BOOTROM_LOCKING_ENABLED=1`），SDK通过`LOCK_ENABLE`启用只读存储器锁定，这两个函数利用其他`SHA_256/FLASH_OP/OTP`只读存储器锁来获取或释放对应只读存储器资源的所有权。

SDK中的`rom_`包装函数在对具备锁定需求的只读存储器函数调用时，会在前后分别调用`bootrom_acquire_lock_blocking`和`bootrom_release_lock`函数。

### 5.4.6. API函数及只读存储器数据的分类列表

每个函数名称后括号中的术语（`Arm-S`、`Arm-NS`、`RISC-V`）指示该API可用的架构及安全状态组合：

- `Arm-S`: 运行于安全状态下的Arm处理器
- `Arm-NS`: 运行于非安全状态下的Arm处理器
- `RISC-V`: RISC-V处理器

有关这些术语的完整定义，请参见第5.4.2节。

以括号结尾的列表项，例如`flash_op()`，表示可调用函数。不带括号的列表项，例如`git_revision`，表示指向只读存储器数据位置的指针。

#### 5.4.6.1 低级Flash访问

这些低级（仅限安全状态）Flash访问函数与RP2040上的类似：

- `connect_internal_flash()` (`Arm-S`、`RISC-V`)
- `flash_enter_cmd_xip()` (`Arm-S`、`RISC-V`)
- `flash_exit_xip()` (`Arm-S`、`RISC-V`)
- `flash_flush_cache()` (`Arm-S`, `RISC-V`)
- `flash_range_erase()` (`Arm-S`, `RISC-V`)
- `flash_range_program()` (`Arm-S`, `RISC`

`-V`) 这是 RP2350 新增的：

- [flash\\_reset\\_address\\_trans\(\)](#) (Arm-S, RISC-V)
- [flash\\_select\\_xip\\_read\\_mode\(\)](#) (Arm-S, RISC-V)

#### 5.4.6.2. 高级闪存访问

更高层次的访问函数，提供在授权许可下对非安全代码暴露的安全功能。

- [flash\\_op\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [flash\\_runtime\\_to\\_storage\\_addr\(\)](#) (Arm-S, Arm-NS, RISC-V)

#### 5.4.6.3. 系统信息

- [flash\\_devinfo16\\_ptr\(\)](#) (Arm-S, RISC-V)
- [get\\_partition\\_table\\_info\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [get\\_sys\\_info\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [git\\_revision\(\)](#) (Arm-S, Arm-NS, RISC-V)

#### 5.4.6.4. 分区表

- [get\\_b\\_partition\(\)](#) (Arm-S, RISC-V)
- [get\\_uf2\\_target\\_partition\(\)](#) (Arm-S, RISC-V)
- [pick\\_ab\\_partition\(\)](#) (Arm-S, RISC-V)
- [partition\\_table\\_ptr\(\)](#) (Arm-S, RISC-V)
- [load\\_partition\\_table\(\)](#) (Arm-S, RISC-V)

#### 5.4.6.5. Bootrom 只读存储器及状态

- [set\\_bootrom\\_stack\(\)](#) (RISC-V)
- [xip\\_setup\\_func\\_ptr\(\)](#) (Arm-S, RISC-V)
- [bootrom\\_state\\_reset\(\)](#) (Arm-S, RISC-V)

#### 5.4.6.6. 可执行镜像管理

- [chain\\_image\(\)](#) (Arm-S, RISC-V)
- [\(explicit\\_buy\(\)\)](#) (Arm-S, RISC-V)

#### 5.4.6.7. 安全性

这些仅限安全的函数用于控制非安全代码的访问权限：

- [set\\_ns\\_api\\_permission\(\)](#) (Arm-S)
- [set\\_rom\\_callback\(\)](#) (Arm-S, RISC-V)
- [validate\\_ns\\_buffer\(\)](#) (Arm-S, RISC-V)

### 5.4.6.8. 其他

这些函数适用于所有平台及安全级别，但当由非安全Arm代码调用时，会执行额外的检查：

- [reboot\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [otp\\_access\(\)](#) (Arm-S, Arm-NS, RISC-V)

### 5.4.6.9. 仅限非安全

- [secure\\_call\(\)](#) (Arm-NS)

### 5.4.6.10. 位操作

与RP2040不同，bootrom不包含位操作函数。RP2350 上的处理器实现了这些操作的硬件指令，其速度远远超过 RP2040 只读存储器中软件实现的版本。

### 5.4.6.11. Memcpy 和 Memset

与 RP2040 不同，只读存储器不提供内存复制或清零函数，因为您的语言运行时预计已在 Cortex-M33 或 Hazard3 上提供了高性能实现。

只读存储器确实包含了标准 C [memcpy\(\)](#) 和 [memset\(\)](#) 的私有实现，适用于 Arm 和 RISC-V，但这些实现优化的是体积而非性能。它们未在只读存储器表中导出。

### 5.4.6.12. 浮点

与 RP2040 不同，只读存储器不包含浮点算术函数。在 Arm 架构上，Cortex-M FPU 提供了对单精度算术的标准处理器支持，RP2350 提供了一个 Arm 协处理器，大幅加速双精度算术（DCP，第 3.6.2 节）。SDK 默认选择性能最佳的硬件或软件实现。

## 5.4.7. API函数及只读存储器数据的字母顺序列表

- [bootrom\\_state\\_reset\(\)](#) (Arm-S, RISC-V)
- [chain\\_image\(\)](#) (Arm-S, RISC-V)
- [connect\\_internal\\_flash\(\)](#) (Arm-S、 RISC-V)
- [flash\\_devinfo16\\_ptr](#) (Arm-S, RISC-V)
- [flash\\_enter\\_cmd\\_xip\(\)](#) (Arm-S、 RISC-V)
- [flash\\_exit\\_xip\(\)](#) (Arm-S、 RISC-V)
- [flash\\_flush\\_cache\(\)](#) (Arm-S, RISC-V)
- [flash\\_op\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [flash\\_range\\_erase\(\)](#) (Arm-S, RISC-V)
- [flash\\_range\\_program\(\)](#) (Arm-S, RISC-V)
- [flash\\_reset\\_address\\_trans\(\)](#) (Arm-S, RISC-V)
- [flash\\_runtime\\_to\\_storage\\_addr\(\)](#) (Arm-S, Arm-NS, RISC-V)

- [flash\\_select\\_xip\\_read\\_mode\(\)](#) (Arm-S, RISC-V)
- [get\\_b\\_partition\(\)](#) (Arm-S, RISC-V)
- [get\\_partition\\_table\\_info\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [get\\_sys\\_info\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [get\\_uf2\\_target\\_partition\(\)](#) (Arm-S, RISC-V)
- [git\\_revision](#) (Arm-S, Arm-NS, RISC-V)
- [load\\_partition\\_table\(\)](#) (Arm-S, RISC-V)
- [otp\\_access\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [partition\\_table\\_ptr](#) (Arm-S, RISC-V)
- [pick\\_ab\\_partition\(\)](#) (Arm-S, RISC-V)
- [reboot\(\)](#) (Arm-S, Arm-NS, RISC-V)
- [secure\\_call\(\)](#) (Arm-NS)
- [set\\_bootrom\\_stack\(\)](#) (RISC-V)
- [set\\_ns\\_api\\_permission\(\)](#) (Arm-S)
- [set\\_rom\\_callback\(\)](#) (Arm-S, RISC-V)
- [validate\\_ns\\_buffer\(\)](#) (Arm-S, RISC-V)
- [xip\\_setup\\_func\\_ptr](#) (Arm-S, RISC-V)

## 5.4.8. API函数清单

### 5.4.8.1. bootrom\_state\_reset

代码: 'S', 'R'

函数签名: void bootrom\_state\_reset(uint32\_t flags)

支持的架构: Arm-S, RISC-V

根据以下标志重置内部只读存储器状态:

- **0x0001 :STATE\_RESET\_CURRENT\_CORE-** 将当前核心的任何内部只读存储器状态重置为已知状态。该方法应在调用当前核心上的其他只读存储器API之前调用，并且在核心0正常启动或核心1代码启动时由只读存储器自动调用。
- **0x0002 :STATE\_RESET\_OTHER\_CORE-** 将另一个核心的所有内部只读存储器状态重置为干净状态。通常由调试器在通过另一个核心上运行的代码重置某个核心状态时调用。
- **0x0004 : STATE\_RESET\_GLOBAL\_STATE -** 重置所有非核心特定状态，包括：
  - 禁用来自Arm-NS的bootrom API访问（另见 set\_ns\_api\_permission()）。
  - 解锁所有启动锁（第5.4.4节）。
  - 清除所有安全代码回调。（另见 set\_rom\_callback()）

请注意，SDK在运行时初始化阶段调用此方法，以将启动只读存储器置于已知状态。此举允许程序在通过调试器进入，或未经过通常通过bootrom的启动路径时，仍能正确运行，因为后者自身会重置状态。

### 5.4.8.2. chain\_image

代码: 'C', 'I'

函数签名: `int chain_image(uint8_t *workarea_base, uint32_t workarea_size, int32_t region_base, uint32_t region_size)`

支持架构: Arm-S, RISC-V。关于RISC-V的说明: 此函数可能需要额外的栈空间; 详见第5.4.8.26节。

返回值: 成功时返回`BOOTROM_OK(0)`, 失败时返回负错误代码。

搜索指定内存区域中的可启动映像, 并在条件允许时执行该映像。

`region_base`和`region_size`指定一个字对齐、大小为字倍数的RAM、XIP RAM或闪存区域以供搜索。

该区域的前4 kB必须包含带有`IMAGE_DEF`的块循环起始部分。若新映像被启动, 该调用不会返回, 否则将返回错误。

`region_base`为有符号值, 可传入负值, 该负值取反后表示既是`region_base`, 也是“闪存更新”区域的基址。

此方法在版本选择与签名校验等方面具备与引导路径类似的复杂性, 因此需要用户提供的内存缓冲区作为工作区。工作区应按字对齐且具备足够大小, 否则将返回`BOOTROM_ERROR_BAD_ALIGNMENT`或`BOOTROM_ERROR_INSUFFICIENT_RESOURCES`错误。当前所需的工作区大小为3064, 因此3 kB是合适的选择。

此方法主要预期用于实现引导加载程序时。

#### ① 注意

在链入映像时, `BOOT_FLAGS0.ROLLBACK_REQUIRED` 标志不会被设置, 以防止通过启动具有回滚版本的二进制文件使无回滚版本的引导加载程序失效 (详见第5.10.8节)。

### 5.4.8.3. connect\_internal\_flash

代码: 'I', 'F'

函数签名: `void connect_internal_flash(void)`

支持的架构: Arm-S, RISC-V

将所有QSPI引脚控制恢复到默认状态, 并将QMI外设连接至QSPI引脚。

如果通过OTP FLASH\_DEVINFO配置了次级闪存芯片选择GPIO, 或通过写入引导RAM中FLASH\_DEVINFO的运行时副本进行了配置, 则该bank 0的GPIO也将被初始化, 且QMI外设被连接。否则, bank 0的IO保持不变。

### 5.4.8.4. explicit\_buy

代码: 'E', 'B'

签名: `int explicit_buy(uint8_t *buffer, uint32_t buffer_size)`

支持的架构: Arm-S RISC-V

返回值: `BOOTROM_OK(0)` 表示成功, 出错时返回负错误代码。

对通过`IMAGE_DEF`启动且带有TBYB (第5.1.17节) 标记的可执行文件执行“显式购买”。对该映像执行“闪存更新”启动是一种仅运行映像一次的方式, 但只有当其通过此调用安全回调引导只读存储器时, 才会成为“当前”映像。

该调用可能执行以下操作:

- 擦除并重写包含TBYB标记的闪存区域以清除此标记。

- 若此新 `IMAGE_DEF` 为版本降级，则在 A/B 分区方案中擦除另一分区的首个扇区，  
(使该映像在非正常启动时能够再次启动)
- 若芯片为安全芯片且映像中包含回滚版本，则更新 OTP 中的回滚版本。

上述第一个示例需要 4 kB 的临时缓冲区，因此在此情况下，应向此方法传递至少 4 kB 的字对齐缓冲区，否则将返回 `BOOTROM_ERROR_BAD_ALIGNMENT /BOOTROM_ERROR_INSUFFICIENT_RESOURCES`。

如果需要写入多个回滚记录行，设备在更新回滚版本时可能会重启。此情况发生于版本号跨越 24 的倍数时（例如，从版本 23 升级到 25 需要重启，但 23 到 24 或 24 到 25 则不需要）。因此，若使用回滚版本，应用程序必须准备在调用此函数时进行重启。

#### 5.4.8.5. `flash_devinfo16_ptr`

代码: 'F', 'D'

类型: `uint16_t *flash_devinfo16_ptr`

指向闪存设备信息的指针，该信息由闪存 API 使用，例如用于对闪存设备大小进行边界检查，以及配置用于次级 QSPI 片选的 GPIO 引脚。

如果 `BOOT_FLAGS0.FLASH_DEVINFO_ENABLE` 被设置，则此启动只读存储器位置将在启动时从 `FLASH_DEVINFO` 初始化，否则将初始化为：

- 芯片选择 0 大小：16 MB
- 芯片选择 1 大小：0 字节
- 无芯片选择 1 GPIO
- 不支持 D8h 擦除命令

闪存 API 使用此启动只读存储器中 `FLASH_DEVINFO` 的副本，因此安全代码可通过该指针在运行时更新闪存设备信息。

#### 5.4.8.6. `flash_enter_cmd_xip`

代码: 'C', 'X'

函数签名: `void flash_enter_cmd_xip(void)`

支持的架构: Arm-S, RISC-V

与 `flash_select_xip_read_mode(0, 12);` 的兼容别名。

配置 QMI 以在每次 XIP 访问时生成标准的 `03h` 串行读取命令，使用 24 位地址。这是较慢的 XIP 配置，但被广泛支持。CLKDIV 设置为 12。调试器可能会调用此函数，以确保程序/擦除操作后闪存可读。

请注意，`flash_exit_xip()` 也执行相同设置，且 RP2350 的闪存程序/擦除函数不会导致 XIP 处于无法访问状态，因此调用此函数通常是多余的。该函数为兼容 RP2040 而提供。

#### 5.4.8.7. `flash_exit_xip`

代码: 'E', 'X'

签名: `void flash_exit_xip(void)`

支持的架构: Arm-S, RISC-V

初始化 QMI 以支持串行操作（直接模式），并初始化基础 XIP 模式，在该模式下，QMI 会以低速（CLKDIV=12）执行 `03h` 串行读取命令以响应 XIP 读取。

然后，向芯片选择 0 上的 QSPI 设备发送一系列指令，旨在使其从连续读取模式（“XIP 模式”）和/或 QPI 模式恢复至可接受串行命令的状态。此步骤在系统复位后必不可少，以将 QSPI 设备恢复至已知状态，因为复位 RP2350 不会复位所连接的 QSPI 设备。当用户代码已执行若干连续读取模式或QPI模式访问后，若需将QSPI设备恢复至能接受引导只读存储器启动程序闪存访问功能发出的串行擦除及编程命令的状态，此操作亦属必要。

若通过`FLASH_DEVINFO`配置了辅助芯片选择的GPIO，则XIP退出序列亦将发送至芯片选择1。

调用此函数后，QSPI设备应可供XIP读取访问；名称`flash_exit_xip`表示将QSPI设备由XIP状态恢复至串行命令状态。

#### 5.4.8.8. `flash_flush_cache`

代码： '`F`'， '`C`'

函数签名： `void flash_flush_cache(void)`

支持的架构： `Arm-S, RISC-V`

通过对每条缓存行执行按组/路维护操作的失效操作（参见第4.4.1节），刷新整个XIP缓存，确保闪存编程与擦除操作对后续缓存的XIP读取生效。

请注意，此操作会解除固定的缓存行，可能会干扰XIP缓存作为SRAM的使用。

不执行其他任何操作。

#### 5.4.8.9. `flash_op`

代码： '`F`'， '`O`'

函数签名： `int flash_op(uint32_t flags, uint32_t addr, uint32_t size_bytes, uint8_t *buf)`

支持的架构： `Arm-S, Arm-NS, RISC-V`

返回值： `BOOTROM_OK(0)` 表示成功，出错时返回负错误代码。

执行闪存读取、擦除或编程操作。擦除操作必须扇区对齐（4096字节）且为扇区大小的整数倍，编程操作必须页对齐（256字节）且为页大小的整数倍；未对齐的擦除和编程操作将返回`BOOTROM_ERROR_BAD_ALIGNMENT`错误。通过`flags`参数中的`CFLASH_OP_BITS`位域选择操作类型——擦除、读取或编程：

`flags` 由以下值组成：

| 地址转换（请选择一项）             |                                                                                                                               |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <code>0x00000000</code> | 无地址转换； <code>addrarg</code> 是实际的闪存存储地址。                                                                                       |
| <code>0x00000001</code> | 运行时地址转换； <code>addrarg</code> 是受地址转换影响的XIP内存地址。                                                                               |
| 安全级别（请选择一项）             |                                                                                                                               |
| <code>0x00000100</code> | 使用安全权限执行操作。非安全调用者禁止执行此操作。                                                                                                     |
| <code>0x00000200</code> | 使用非安全权限执行操作。                                                                                                                  |
| <code>0x00000300</code> | 使用引导加载程序权限执行操作。非安全调用者禁止执行此操作。                                                                                                 |
| 操作（请选择一项）               |                                                                                                                               |
| <code>0x00000000</code> | 从地址 <code>addr</code> 开始擦除 <code>size_bytes</code> 字节的闪存。 <code>addr</code> 和 <code>size_bytes</code> 必须均为4096字节（一个闪存扇区）的整数倍。 |

|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| <code>0x00010000</code> | 从地址addr开始写入 size_bytes字节的闪存。addr和size_bytes必须均为256字节（一个闪存页）的整数倍。 |
| <code>0x00020000</code> | 读取 size_bytes 字节闪存，起始地址为 addr 。 地址 addr 和字节数 size_bytes 均无对齐限制。  |

这些常量在 SDK 的 `bootrom_constants.h` 中也以 `#define` 形式提供。

地址 `addr` 是访问的第一个闪存字节地址，范围包括从 `XIP_BASE` 至 `XIP_BASE + 0x1fffff`。该地址可能为运行时地址或存储地址。缓冲区 `buf` 用于存放编程操作写入闪存的数据，及读取操作时从闪存回读的数据。缓冲区 `buf` 从不被编程操作写入，擦除操作完全忽略该缓冲区。

闪存操作会针对保存在启动 RAM 中的 `FLASH_DEVINFO` 运行时值所指定的已知闪存设备执行边界检查。若 `BOOT_FLAGS0.FLASH_DEVINFO_ENABLE` 被设置，则该值由启动只读存储器根据 OTP 中的 `FLASH_DEVINFO` 初始化；否则，芯片选择 0 初始化为 16 MB，芯片选择 1 初始化为 0 字节。`FLASH_DEVINFO` 可通过写入其在引导RAM中的位置来在运行时更新，该位置的指针可在只读存储器表中查找。

如果启用了常驻分区表，则闪存操作还需依据分区权限进行校验。本函数的安全版本可通过 `flags` 参数中的 `CFLASH_SECLEVEL_BITS` 位域指定调用者的有效安全级别（安全、非安全、引导加载程序），而非安全版本函数始终依据分区的非安全权限进行校验。不允许跨越两个分区的闪存操作，且此类操作将在地址验证时失败。

若 `FLASH_DEVINFO.D8H_ERASE_SUPPORTED` 被设置，则擦除操作将在可能范围内使用 D8h 64 kB 块擦除命令（不超出指定区域），以加快擦除速度。否则，仅使用 20h 4 kB 扇区擦除命令。

该API可根据当前由 QMI 地址转换寄存器 ATRANS0 至 ATRANS7 配置的转换，将 `addr` 从闪存运行时地址转换为闪存存储地址。例如，映像存储于闪存的 +2 MB 偏移处（运行时映射为 XIP 地址 0），对映像偏移 +1 MB 处写入时，实际写入物理闪存存储地址为 3 MB。通过设置 `flags` 参数中的相应位域启用此转换。

启用翻译时，跨越XIP运行时地址空间中地址空洞（由非最大`ATRANSx_SIZE`创建）的闪存操作将返回错误响应。此检查可能导致传输中断：传输可能在遇到地址空洞前部分完成，最终返回失败。

启用翻译时，允许闪存操作跨越芯片选择边界，前提是不跨越ATRANS地址空洞。禁用翻译时，整个操作必须针对单个闪存芯片选择（由地址第24位及以上位决定），否则地址验证将失败。

从安全代码在运行时地址空间中擦除闪存扇区的典型调用序列如下：

- `connect_internal_flash();`
- `flash_exit_xip();`
- `flash_op((CFLASH_OP_VALUE_ERASE << CFLASH_OP_LSB) | (CFLASH_SECLEVEL_VALUE_SECURE << CFLASH_SECLEVEL_LSB) | (CFLASH_ASPACE_VALUE_RUNTIME << CFLASH_ASPACE_LSB), addr, 4096, NULL);`
- `flash_flush_cache();`
- 将 XIP 设置函数从启动只读存储器复制到 SRAM 并执行，以恢复原始的 XIP 模式。
  - bootrom 会写入一个默认的设置函数，用以恢复闪存搜索期间检测到的 mode/clkdiv 参数；用户代码可以用自定义的设置函数覆盖该默认函数。

程序操作亦需类似流程。读取操作可保持当前 XIP 模式有效，因此仅需执行`flash_op(...);`调用。

请注意，RP2350 bootrom 在程序／擦除操作间将闪存保持在基础的 XIP 状态。但在程序／擦除操作期间，QMI 处于直接模式（详见第 12.14.5 节），任何 XIP 访问尝试均会返回

总线错误响应。

#### 5.4.8.10. `flash_range_erase`

代码: 'R', 'E'

函数签名: `void flash_range_erase(uint32_t addr, size_t count, uint32_t block_size, uint8_t block_cmd)`

支持的架构: Arm-S, RISC-V

擦除 `count`字节，从 `addr` (闪存起始位置偏移) 开始。可选传入块擦除命令 (例如, `D8h`块擦除) 及该命令擦除的块大小 – 该函数将在可能情况下使用更大块擦除，从而大幅提升擦除速度。`addr`必须对齐至4096字节扇区，且 `count`必须为4096字节的整数倍。

这是低级闪存API，不对参数进行任何有效性校验。详见 `flash_op()`，该高级API会校验对齐、闪存边界及分区权限，并能透明地实现运行时地址到存储地址的转换。

调用此API前，QSPI设备必须处于串行命令状态，通常通过依次调用 `connect_internal_flash()` 及 `flash_exit_xip()` 实现。擦除完成后，应调用 `flash_flush_cache()` 刷新闪存缓存，以确保修改后的闪存数据对缓存的XIP访问可见。

最后，应通过将保存在引导RAM中的XIP设置函数复制回SRAM并执行该函数来恢复原始XIP模式：`bootrom`提供一个默认函数，用于恢复闪存扫描期间检测到的闪存模式和时钟分频，用户程序可通过自定义XIP设置函数覆盖该默认设置。

在擦除操作期间，QMI处于直接模式（见第12.14.5节），尝试通过DMA、调试器或另一核心访问XIP将导致总线错误。函数执行返回后，XIP将重新可访问。

#### 5.4.8.11. `flash_range_program`

代码: 'R', 'P'

函数签名: `void flash_range_program(uint32_t addr, const uint8_t *data, size_t count)`

支持的架构: Arm-S, RISC-V

向从 `addr` (相对于闪存起点的偏移地址) 开始的闪存存储区域编程数据，大小为 `count`字节。`addr`必须按256字节边界对齐，`count`必须为256字节的整数倍。

这是低级闪存API，不对参数进行任何有效性校验。详见 `flash_op()`，该高级API会校验对齐、闪存边界及分区权限，并能透明地实现运行时地址到存储地址的转换。

调用此API之前，QSPI设备必须处于串行命令状态——详见 `flash_range_erase()` 注释。

#### 5.4.8.12. `flash_reset_address_trans`

代码: 'R', 'A'

函数签名: `void flash_reset_address_trans(void)`

支持的架构: Arm-S, RISC-V

将QMI地址转换寄存器ATRANS0至ATRANS7恢复至复位状态。此操作使运行时到存储地址映射成为恒等映射，即映射地址与未映射地址相等，且整个地址空间完全映射。详见第12.14.4节。

#### 5.4.8.13. `flash_runtime_to_storage_addr`

代码: 'F', 'A'

函数签名: `int flash_runtime_to_storage_addr(uint32_t addr)`

支持的架构: `Arm-S, Arm-NS, RISC-V`

返回值: 成功时为正数（转换后的地址），失败时为负错误码

应用当前由QMI地址转换寄存器ATRANS0至ATRANS7配置的地址转换。

详见第12.14.4节。

将地址转换到XIP运行时地址窗口之外，或超出`ATRANSx_SIZE`字段范围时，返回`BOOTROM_ERROR_INVALID_ADDRESS`，该值不是有效的闪存存储地址。否则，返回QMI在给定运行时地址 `addr`时实际访问的存储地址。这实质上是针对QMI的虚拟地址到物理地址的转换。

#### 5.4.8.14. `flash_select_xip_read_mode`

代码: 'X', 'M'

签名: `void flash_select_xip_read_mode(bootrom_xip_mode_t mode, uint8_t clkdiv)`

支持的架构: `Arm-S, RISC-V`

配置QMI以支持bootrom提供的一组有限XIP读取模式之一。该模式同时配置两个内存窗口（两个芯片选择），且时钟分频也适用于直通模式。

可用模式如下：

- **0**: 03h 串行读取：串行地址，串行数据，无等待周期
- **1**: 0Bh 串行读取：串行地址，串行数据，8周期等待
- **2**: BBh 双IO读取：双地址、双数据，4个等待周期（包括驱动为0的MODE位）
- **3**: EBh 四IO读取：四地址、四数据，6个等待周期（包括驱动为0的MODE位）

XIP写入命令/格式不由此函数配置。

从闪存启动时，bootrom会依次尝试这些模式，从3向0递减。首次检测到可用模式后，会将其记录，并在启动RAM中写入一个调用此函数的默认XIP设置函数，

(`flash_select_xip_read_mode`)，参数为闪存扫描过程中发现的配置。此函数可随时调用，以恢复闪存启动时发现的闪存参数。

所有由bootrom配置的XIP模式均带8位串行命令前缀，使闪存设备能够保持在串行命令状态，从而更灵活地混合进行XIP访问与程序/擦除串行操作。此操作存在性能损失，因此用户可以在闪存启动后，通过使用连续读取模式或QPI模式自行完成闪存配置，以避免或减轻命令前缀带来的开销。

#### 5.4.8.15. `get_b_partition`

代码: 'G', 'B'

函数签名: `int get_b_partition(uint partition_a)`

支持的架构: `Arm-S RISC-V`

返回值: 如果存在且已加载分区表，且分区A存在对应的分区B，则返回分区A对应的分区B索引；否则返回`BOOTROM_ERROR_NOT_FOUND`。

### 5.4.8.16. `get_partition_table_info`

代码： 'G', 'P'

函数签名： `int get_partition_table_info(uint32_t *out_buffer, uint32_t out_buffer_word_size, uint32_t flags_and_partition)`

支持的架构： Arm-S, Arm-NS, RISC-V

返回值： 成功时返回非负值（填充至`out_buffer`中的字数），失败时返回负错误码。

用分区表信息填充缓冲区。请注意，该API也用于通过PICOBOOT接口返回信息。

成功时，缓冲区将被填充，并返回填充到缓冲区中的字数。如果分区表尚未加载（例如，从看门狗或RAM引导），此方法将返回`BOOTROM_ERROR_PRECONDITION_NOT_MET`，您应首先通过`load_partition_table()`函数加载分区表。

由于大小限制，bootrom并未将所有分区表数据常驻内存。为了防止bootrom加载常驻部分后闪存被修改，bootrom会保存加载时分区表的哈希值。如果调用此方法时哈希值已发生改变，则会返回`BOOTROM_ERROR_INVALID_STATE`。

返回的信息由`flags_and_partition`参数决定；返回缓冲区中的第一个字，为该API所支持标志的（子）集合。在解释缓冲区内容前，您应始终检查该值。

继第一个字之后，依序返回每个已置位标志对应的数据字。除`PT_INFO`外，所有标志均选择“每分区”信息，因此每个字段按标志顺序逐一返回各分区数据。特殊的`SINGLE_PARTITION`标志表示仅需要单个分区数据。标志包括：

- `0x0001 - PT_INFO`: 关于整个分区表的整体信息。未分区空间的后两个字，其格式同第 5.9.4.2 节所述。
  - 字 0 : `partition_count` (低 8 位) , `partition_table_present` (第 8 位)
  - 字 1 : `unpartitioned_space_permissions_and_location`
  - 字 2 : `unpartitioned_space_permissions_and_flags`
- `0x8000 -SINGLE_PARTITION`: 仅返回单个分区数据；分区号存储于`flags_and_partition`的高 8 位。

每个分区字段：

- `0x0010 -PARTITION_LOCATION_AND_FLAGS`: 分区的核心信息。这些字段的格式详见第5.9.4.2节。
  - 字0 -`permissions_and_location`
  - 字1 -`permissions_and_flags`
- `0x0020 -PARTITION_ID`: 分区的可选64位标识符。如果在分区标志中设置了`HAS_ID`位，则返回64位ID：
  - 字0 - 前32位
  - 字1 - 后32位
- `0x0040 -PARTITION_FAMILY_IDS`: 分区支持通过MSD引导加载程序下载的除权限和标志字段中标记的标准ID外的任何额外UF2家族ID（详见第5.9.4.2节）。
- `0x0080 -PARTITION_NAME`: 分区的可选名称。如果`permissions_and_flags`中的`HAS_NAME`字段位未设置，则该分区不返回任何数据；否则格式如下：
  - 字节 0 : 名称的7位长度 (LEN) ; 最高位保留
  - 字节 1 : 名称的第一个字符
  - ...

- 字节 LEN：名称的最后一个字符
- ... (填充至下一个字边界)

### 注意

未分区空间始终在字 1 中报告，基准偏移为 `0x0`，大小为 `0x2000`扇区（32 MB）。只读存储器引导程序将未分区空间权限应用于任何未被分区覆盖的闪存存储地址。

#### 5.4.8.17. `get_sys_info`

代码: '`G`', '`S`'

函数签名: `int get_sys_info(uint32_t *out_buffer, uint32_t out_buffer_word_size, uint32_t flags)`

支持的架构: `Arm-S`, `Arm-NS`, `RISC-V`

返回值: 成功时返回正值（填充到 `out_buffer` 的字数），失败时返回负错误代码。

填充缓冲区以包含各种系统信息。请注意，该API也用于通过PICOBOOT接口返回信息。

返回的信息由 `flags` 参数决定；返回缓冲区中的第一个字，为该API所支持标志的（子）集合。在解释缓冲区内容前，您应始终检查该值。

继第一个单词后，按顺序返回每个已设置标志对应的数据单词：

- `0x0001 : CHIP_INFO` - 芯片的唯一标识符（3 个单词）
  - 单词 0: `CHIP_INFO_PACKAGE_SEL` 寄存器的值
  - 单词 1: RP2350 设备 ID 低位
  - 单词 2: RP2350 设备 ID 高位
- `0x0002 : CRITICAL` (1 个单词)
  - 单词 0: OTP CRITICAL 寄存器的值，包含上一次 OTP 复位事件读取的关键启动标志
- `0x0004 : CPU_INFO` (1 个单词)
  - 单词 0: 当前 CPU 架构
    - 0 - Arm
    - 1 - RISC-V
- `0x0008 : FLASH_DEV_INFO` (1 个单词)
  - 单词 0: 以 OTP FLASH\_DEVINFO 格式表示的闪存设备信息
- `0x0010 : BOOT_RANDOM` - 每次启动时生成的 128 位随机数（4 个单词）
  - Word 0: 每次启动的随机数 0
  - Word 1: 每次启动的随机数 1
  - Word 2: 每次启动的随机数 2
  - Word 3: 每次启动的随机数 3
- `0x0020 : NONCE` - 不支持
- `0x0040 : BOOT_INFO` (4 个字)
  - Word 0: `0xffffffff`
    - `tt` - 最近启动的 TBYB 及更新信息（在常规非 BOOTSEL 启动时更新）

- **pp** - 最近启动的分区（在常规非 BOOTSEL 启动时更新）
- **bb** - 最近启动的启动类型
- **dd** - 最近启动的诊断“分区”
- Word 1 : 最近启动的诊断信息。来自最近一次启动的诊断信息（含上述 dd 指示的分区或槽的信息）。
  - “分区”编号如下：
  - 0-15 : 分区编号
  - -1 : 无
  - -2 : 槽 0
  - -3 : 槽 1
  - -4 : 镜像（诊断来源于RAM镜像启动、OTP引导镜像或用户chain\_image()调用）。
- 字 2 : 上次重启参数 0
- 字 3 : 上次重启参数 1

“引导诊断”信息用于帮助识别启动失败的原因，或启动到意外二进制文件的情况。此信息可通过PICOBOOT在看门狗重启后获取，但在通过RUN引脚复位或POWMAN复位时不会保留。

诊断信息仅包含一个字。其记录内容基于上述 pp选择，后者作为编程方式重新启动至正常引导时的参数。

要获取诊断信息，**pp**必须指向一个槽或“A”分区；镜像诊断会在从OTP或RAM镜像引导时自动选择，或当chain\_image()被调用时自动选择。

因此，诊断字包含槽0和槽1的数据，或“A”分区及其存在的“B”分区的数据。  
诊断字的低半字包含来自槽0或分区A的信息；高半字包含来自槽1或分区B的信息。

每个半字的格式如下（使用单词 *region*指代槽或分区）：

- **0x0001 :REGION\_SEARCHED** - 该区域已搜索块循环。
- **0x0002 :INVALID\_BLOCK\_LOOP** - 发现了块循环，但该循环无效。
- **0x0004 :VALID\_BLOCK\_LOOP** - 发现了有效的块循环（由完全包含于该区域的循环中的块组成，且这些块具有正确的结构。每个块由大小总和等于块大小的项组成）。
- **0x0008 :VALID\_IMAGE\_DEF** - 该区域发现了有效的 IMAGE\_DEF。有效的 IMAGE\_DEF 必须能够正确解析且必须可执行。
- **0x0010 :HAS\_PARTITION\_TABLE** - 是否存在分区表。如若设置了 VALID\_BLOCK\_LOOP，分区表必须具备正确结构。若分区表被判定为无效，则INVALID\_BLOCK\_LOOP亦将设置（因而VALID\_BLOCK\_LOOP与INVALID\_BLOCK\_LOOP均被设置）。
- **0x0020 :CONSIDERED** - 已考虑该分区/槽的选择。第一个槽/分区的选择依据多项因素确定。若首选未通过验证，则考虑另一选项。
  - 槽/分区中所包含的PARTITION\_TABLE/ IMAGE\_DEF 版本信息。
  - 槽/分区是否作为FLASH\_UPDATE重启时的“更新区域”。
  - 是否将 IMAGE\_DEF 标记为“明确购买”。
- **0x0040 :CHosen** - 此槽/分区已选定（或为唯一选择）。
- **0x0080 :PARTITION\_TABLE\_MATCHING\_KEY\_FOR\_VERIFY** - 若PARTITION\_TABLE需签名以验证（通过OTP设置），则表明该PARTITION\_TABLE是否使用匹配OTP中四个密钥之一的密钥进行了签名。
- **0x0100 :PARTITION\_TABLE\_HASH\_FOR\_VERIFY** - 如果可以执行哈希值校验，则设置此值。如果需要签名，则此值与PARTITION\_TABLE\_MATCHING\_KEY\_FOR\_VERIFY相同。

- 0x0200 :PARTITION\_TABLE\_VERIFIED\_OK- 表示PARTITION\_TABLE是否通过校验（如存在或必需，包含签名/哈希）。
- 0x0400 :IMAGE\_DEF\_MATCHING\_KEY\_FOR\_VERIFY - 若因安全启动要求为 IMAGE\_DEF 签名，则表示 IMAGE\_DEF 是否使用与OTP中存储的四个密钥之一匹配的密钥进行了签名。
- 0x0800 :IMAGE\_DEF\_HASH\_FOR\_VERIFY- 如果可以执行哈希值校验，则设置此值。如果需要签名，则此值与IMAGE\_DEF\_MATCHING\_KEY\_FOR\_VERIFY相同。
- 0x1000 :IMAGE\_DEF\_VERIFIED\_OK- 表示PARTITION\_TABLE是否通过校验（如存在或必需，包含签名/哈希），且任何 LOAD\_MAP 是否有效。
- 0x2000 :LOAD\_MAP\_ENTRIES\_LOADED- 是否因 LOAD\_MAP 将任何代码复制到RAM中
- 0x4000 :IMAGE\_LAUNCHED- 是否启动了该区域的 IMAGE\_DEF
- 0x8000 :IMAGE\_CONDITION\_FAILURE - 启动前 IMAGE\_DEF 未通过最终检查；这些检查包括：
  - 验证失败（如果之前在 CONSIDERED 阶段未验证）。
  - 设置任何滚动窗口时发生问题。
  - 在OTP中无法设置回滚版本（如安全模式要求）
  - 映像被标记为非安全
  - 映像被标记为“explicit buy”，且此次为闪存启动，但该区域并非“闪存更新”区域
  - 映像架构错误，但架构自动切换已禁用（或正确架构已禁用）

### **i 注意**

同时设置BOOT\_DIAGNOSTIC\_INVALID\_BLOCK\_LOOP和BOOT\_DIAGNOSTIC\_VALID\_BLOCK\_LOOP的矛盾组合用于标识一个PARTITION\_TABLE，该表在初步验证（包括散列/签名）后通过，但完全解析时发现内容无效。

为全面了解涉及槽位和多个分区的启动失败情况，设备可多次重启以收集相关信息。

#### 5.4.8.18. `get_uf2_target_partition`

代码: 'G', 'U'

签名: `int get_uf2_target_partition(uint8_t *workarea_base, uint32_t workarea_size, uint32_t family_id, resident_partition_t *partition_out)`

支持的架构: Arm-S RISC-V. 关于 RISC-V 的说明：此函数需要额外的栈空间；详见第5.4.8.26节。

返回值：成功时返回  $\geq 0$ （目标分区索引），失败时返回负错误代码。

该方法执行与在 BOOTSEL 模式下将 UF2 文件拖入 USB 驱动器时相同的操作，以确定 UF2 家族 ID 的目标分区。

此方法在版本选择与签名校验等方面具备与引导路径类似的复杂性，因此需要用户提供的内存缓冲区作为工作区。工作区应按字节和字对齐且大小充足，否则将返回BOOTROM\_ERROR\_INSUFFICIENT\_RESOURCES 错误。当前所需的工作区大小为3064，因此3K是较为合适的选择。

如果尚未加载分区表（例如，在看门狗或RAM启动时），该方法将返回BOOTROM\_ERROR\_PRECONDITION\_NOT\_MET，您应先通过load\_partition\_table()加载分区表。

#### 5.4.8.19. git\_revision

代码: 'G', 'R'

类型:const uint32\_t git\_revision

Bootrom git修订版本的8位最高有效十六进制数字。唯一标识此版本的bootrom。

##### **i 注意**

该git修订版本为芯片版图确认时构建；由于历史被压缩，公共仓库中的git哈希不同，尽管内容一致。可通过构建公共bootrom源代码并将生成的二进制文件与芯片导出的二进制文件进行比对来验证内容一致性。

#### 5.4.8.20. load\_partition\_table

代码: 'L', 'P'

函数签名: int load\_partition\_table(uint8\_t \*workarea\_base, uint32\_t workarea\_size, bool force\_reload)

支持架构: Arm-S, RISC-V。关于 RISC-V 的说明: 此函数需要额外的栈空间；详见第5.4.8.26节。

返回值: 成功时返回 BOOTROM\_OK(0)，失败时返回负错误代码。

从闪存加载当前分区表（如果存在）。

此方法在版本选择与签名校验等方面具备与引导路径类似的复杂性，因此需要用户提供的内存缓冲区作为工作区。工作区域应按字节和字对齐且大小充足，否则将返回 BOOTROM\_ERROR\_INSUFFICIENT\_RESOURCES 错误。当前所需的工作区大小为 3064，因此 3K 是较为合适的选择。

如果 force\_reload 为假，则当 bootrom 已加载时本方法将立即返回 BOOTROM\_OK；否则，如果分区表已加载，将重新加载分区表，以允许在运行程序中更新分区表。

#### 5.4.8.21. otp\_access

代码: 'O', 'A'

函数签名: int otp\_access(uint8\_t \*buf, uint32\_t buf\_len, uint32\_t row\_and\_flags)

支持的架构: Arm-S, Arm-NS, RISC-V

返回值: 成功时返回 BOOTROM\_OK(0)，失败时返回负错误代码。

将缓冲区中的数据写入 OTP，或从 OTP 读取数据至缓冲区。

- 0x0000ffff - ROW\_NUMBER: 低16位为行号 (0-4095)
- 0x00010000 - IS\_WRITE: 设置此位时执行写操作 (非读操作)
- 0x00020000 - IS\_ECC: 若该位被设置，缓冲区中每个值为2字节，操作时针对OTP中24位值采用ECC；若未设置，缓冲区中每个值为4字节，其中低24位被写入或从OTP读取。

缓冲区必须根据 IS\_ECC 标志对齐至 2 字节或 4 字节。

该方法将持续读取和写入行，直到遇到首个未通过键值或权限检查的行，此时将返回 BOOTROM\_ERROR\_NOT\_PERMITTED。

写入操作也将在首次尝试将 OTP 位从 1 更改为 0 的行时停止，并返回 BOOTROM\_ERROR\_UNSUPPORTED\_MODIFICATION。

若所有行均成功读取/写入，则返回 BOOTROM\_OK。

#### 5.4.8.22. `partition_table_ptr`

代码: 'P', 'T'

类型: `resident_partition_table **partition_table_ptr`

指向常驻分区表信息指针的指针。常驻分区表是完整分区表的子集，驻留于内存中，用于闪存权限控制。

常驻分区表信息的公开部分格式如下：

| 字     | 字节    | 值                                                                                                                    |
|-------|-------|----------------------------------------------------------------------------------------------------------------------|
| 0     | 1     | <code>partition_count (0-16)</code>                                                                                  |
|       | 1     | <code>partition_count_with_permissions (0-16)</code> 。运行时增加额外权限区域时，应将此值设置为大于 <code>partition_count</code> （请勿修改原有分区） |
|       | 1     | <code>loaded (0x01 表示已从闪存加载分区表)</code>                                                                               |
|       | 1     | <code>0x00</code> （填充）                                                                                               |
| 1     | 1     | 未分区空间权限及标志                                                                                                           |
| 2-3   | 分区 0  |                                                                                                                      |
|       | 1     | 分区 0 的权限与位置                                                                                                          |
|       | 1     | 分区 0 的权限及标志                                                                                                          |
| 4-5   | 分区 1  |                                                                                                                      |
|       | 1     | 分区 1 的权限与位置                                                                                                          |
|       | 1     | 分区 1 的权限及标志                                                                                                          |
| ...   | ...   | ...                                                                                                                  |
| 32-33 | 分区 15 |                                                                                                                      |
|       | 1     | 分区 15 的权限与位置                                                                                                         |
|       | 1     | 分区 15 的权限及标志                                                                                                         |

字段`permissions_and_location`及`permissions_and_flags`的详细信息见第 5.9.4 节。

#### 5.4.8.23. `pick_ab_partition`

代码: 'A', 'B'

函数签名: `int pick_ab_partition(uint8_t *workarea_base, uint32_t workarea_size, uint partition_a_num)`

支持架构: Arm-S, RISC-V。关于 RISC-V 的说明: 此函数需要额外的栈空间; 详见第 5.4.8.26 节。

返回值: 成功时返回  $\geq 0$  (分区索引)，失败时返回负错误代码。

确定哪个分区具有“更优”的 `IMAGE_DEF`。对于可执行映像，此分区将被引导启动。

此方法在版本选择与签名校验等方面具备与引导路径类似的复杂性，因此需要用户提供的内存缓冲区作为工作区。工作区应为字对齐且大小足够，否则将返回`BOOTROM_ERROR_INSUFFICIENT_RESOURCES`错误。当前所需的工作区大小为 3064，因此 3K 是较为合适的选择。

传入的分区编号可为除 A/B 对中“B”分区之外的任何有效分区编号。

本方法返回负错误代码，或选中分区的分区编号（即`partition_a_num`或其对应的“B”分区编号）。

### ⓘ 注意

本方法不会检查所有者分区，仅对传入的 A 分区及其对应 B 分区进行判断。

#### 5.4.8.24. reboot

代码: 'R', 'B'

函数签名: `int reboot(uint32_t flags, uint32_t delay_ms, uint32_t p0, uint32_t p1)`

支持的架构: Arm-S, Arm-NS, RISC-V

返回值: `BOOTROM_OK` (或不返回) 表示成功，出错时返回负错误码。

复位RP2350并使用看门狗功能重启。

参数 `delay_ms` 是重启前的毫秒延迟时间。注意：默认情况下，此方法为异步（除非设置了`NO_RETURN_ON_SUCCESS`—详见下文），方法会立即返回，重启将在延迟的毫秒数后发生。

参数 `flags` 字段包含以下值之一：

- `0x0000 : REBOOT_TYPE_NORMAL` - 重新启动至正常启动路径。
  - `p0` - 引导诊断“分区”（仅限低8位）
- `0x0002 : REBOOT_TYPE_BOOTSEL` - 重新启动至BOOTSEL模式。
  - `p0` - 一组标志：
    - `0x01 : DISABLE_MSD_INTERFACE` - 禁用BOOTSEL USB驱动（详见第5.5节）
    - `0x02 : DISABLE_PICOBOOT_INTERFACE` - 禁用 PICOBLOCK 接口（参见第 5.6 节）。
    - `0x10 : GPIO_PIN_ACTIVE_LOW` - GPIO 在 `p1` 为低电平有效。
    - `0x20 : GPIO_PIN_ENABLED` - 启用指定 GPIO 上的活动指示灯。
  - `p1` - 用作 BOOTSEL USB 驱动器活动指示灯的 GPIO 编号（由 `p0` 中的标志启用）。
- `0x0003 :REBOOT_TYPE_RAM_IMAGE` - 重启至 RAM 中的映像。系统将搜索 RAM 或 XIP RAM 区域以运行映像。此类重启用于将 RAM UF2 文件拖拽至 BOOTSEL USB 驱动器时。
  - `p0` - 区域起始地址（字对齐）。
  - `p1` - 区域大小（字对齐）。
- `0x0004 :REBOOT_TYPE_FLASH_UPDATE`- `REBOOT_TYPE_NORMAL` 的变体，用于闪存更新后重启。这是在将闪存UF2文件拖放到 `BOOTSEL` USB 驱动器后使用的重启类型。
  - `p0`- 已更新闪存区域的起始地址。如果该地址与分区或槽的起始地址相符，则在启动（有多个选择时）期间，该分区或槽将被优先处理。
  - 此类启动支持 TBYB（第5.1.17节）及版本降级。
- `0x000d :REBOOT_TYPE_PC_SP`- 重启到指定的 PC 和 SP。注意：Arm-NS 变体中禁止使用此项。
  - `p0`- 用于开始执行的初始程序计数器（PC）。对 Arm 架构而言，最低位必须置位；对 RISC-V 架构而言，最低位必须清零。
  - `p1` - 初始堆栈指针（SP）。

上述所有项，都可以选择性地按位或（OR）以添加下列标志：

- `0x0010 :REBOOT_TO_ARM`- 将两个核心均切换至 Arm 架构（而非保持当前状态）。如果不支持 Arm 架构，调用将失败，返回`BOOTROM_ERROR_INVALID_STATE`错误。
- `0x0020 :REBOOT_TO_RISCV`- 将两个核心均切换至 RISC-V 架构（而非保持原状）。如果不支持 RISC-V 架构，调用将失败，返回`BOOTROM_ERROR_INVALID_STATE`错误。

- `0x0100 :NO_RETURN_ON_SUCCESS`— 看门狗硬件为异步模式。设置此位将在成功启动重启后强制方法不返回。

### 注意

参数 `p0` 和 `p1` 通常写入看门狗暂存寄存器 2 和 3，由启动路径代码在重启后进行解释。例外情况为 `REBOOT_TYPE_NORMAL`，调用此 API 会在重启前处理 `p0` 值；启动路径自身则不接受任何 `REBOOT_TYPE_NORMAL` 参数。

#### 5.4.8.25. `secure_call`

代码: '`S`' , '`C`'

签名: `int secure_call(...)`

支持的架构: `Arm-NS`

返回值: 成功时返回  $\geq 0$ ，出错时返回负数错误代码。

从非安全代码调用安全方法，将待调用的方法传递至寄存器 `r4`（其他参数按常规传递）。

该方法提供了将非安全代码与安全代码解耦的能力，使前者能够调用后者的方法，而无需知晓方法的具体位置。

此调用将始终返回 `BOOTROM_ERROR_INVALID_STATE`，除非安全 Arm 代码已通过 `set_rom_callback()` 提供了处理函数；若存在处理函数，该方法将返回该处理函数的返回码，约定如果“函数选择器”（位于 `r4`）不被支持，则返回 `BOOTROM_ERROR_INVALID_ARG`。

某些已知的“函数选择器”将被预先定义，以便促进安全与非安全 SDK 代码之间，或与其他环境（例如，向安全 UART/USB CDC 记录日志、从非安全代码启动核心 1、从非安全代码通过看门狗重启返回非安全代码等）的交互。为避免冲突，以下位模式将被用于“函数选择器”：

- `0b0xxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` 是一个“众所周知”的函数选择器；请勿用于您自己的方法
- `0b10xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` 是一个“唯一”的函数选择器，旨在避免与其他选择器冲突。低 30 位应随机选择
- `0b11xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx` 是一个“私有”的函数选择器，仅供紧密耦合的非安全(NS)与安全(S)代码使用

#### 5.4.8.26. `set_bootrom_stack`

代码: '`S`' , '`S`'

签名: `int set_bootrom_stack(uint32_t base_size[2])`

支持的架构: `RISC-V`

返回值: `BOOTROM_OK(0)` 表示成功，负数错误代码表示失败。

大多数 bootrom 函数仅使用 Arm 代码编写一次，以节省空间。因此，在 RISC-V 架构下运行时，这些函数须模拟执行。这对用户基本透明，但 Arm 模拟使用的堆栈与调用用户的堆栈分开，存储于 boot RAM 中，且容量有限。在使用某些较复杂的 API 或从 IRQ 内嵌套调用 bootrom 时，可能需要分配更大的栈空间。

此方法允许调用者通过传入指向两个值的指针，指定 RAM 区域作为当前核心的栈使用：对齐的基地址和大小（以字节为单位，4 的倍数）。

该方法在返回前，会将之前的基地址和大小值写入传入的数组中。

### 5.4.8.27. set\_ns\_api\_permission

代码: 'S', 'P'

函数签名: `int set_ns_api_permission(uint ns_api_num, bool allowed)`

支持架构: Arm-S

返回值: `BOOTROM_OK(0)` 表示成功, 负数错误代码表示失败。

允许或禁止特定的非安全 (NS) API; 所有非安全API默认均为禁用。

`ns_api_num` 为以下之一, 用于配置 Arm-NS 对指定API的访问权限。禁用状态的非安全API调用将返回`BOOTROM_ERROR_NOT_PERMITTED`。

- 0x0: `get_sys_info`
- 0x1: `flash_op`
- 0x2: `flash_runtime_to_storage_addr`
- 0x3: `get_partition_table_info`
- 0x4: `secure_call`
- 0x5: `otp_access`
- 0x6: `reboot`
- 0x7: `get_b_partition`

#### ① 注意

重置后, 所有权限默认均被拒绝 (详见 `bootrom_state_reset()`)。

### 5.4.8.28. set\_rom\_callback

代码: 'R', 'C'

函数签名: `int set_rom_callback(uint callback_number, int (*callback)(…))`

支持的架构: Arm-S, RISC-V

返回值: 成功时返回 `>=0` (旧的回调指针), 失败时返回负错误码。

当前唯一支持的`callback_number` 是 0, 用于设置`secure_call` 接口的回调函数。

回调指针为 0 时删除回调函数; 正值回调指针 (所有有效函数指针均在 RP2350 上) 设置回调函数; 负值回调指针则用于获取旧回调指针而不进行更改。

如果成功, 返回值 `>=0` (函数入口时函数指针的现有值)。

### 5.4.8.29. validate\_ns\_buffer

代码: 'V', 'B'

签名: `void *validate_ns_buffer(const void *addr, uint32_t size, uint32_t write, uint32_t *ok)`

支持架构: Arm-S

返回值: `addr` 成功时返回, 错误时返回负错误码。在 RP2350 A3 及更新版本中, 存在额外的带外返回值, 详见下文。

此工具方法可供安全 Arm 代码用于验证从非安全代码传入的缓冲区。

`write` 参数及（输出）结果参数 `ok` 均为 RCP 布尔值，真值为 `0xa500a500`，假值为 `0x00c300c3`。此举增强了该函数的加固性，且 `write` 参数必须为上述值之一，否则 RCP 将终止系统运行。

为确保成功，整个缓冲区必须位于范围 `XIP_BASE` → `SRAM_END` 内，且必须依据 SAU 与 NS MPU 的配置（基于当前处理器的 IP SR 和 NS CONTROL 标志判断是否特权）对非安全调用者开放访问。若通过 ACCESSCTRL 授权，USB RAM 中的缓冲区亦允许非安全访问。

在 RP2350 A3 及更高版本中，该函数会返回额外的带外信息，以支持汇编代码的调用：

- 返回的 RCP 布尔值不仅存放在由 `ok` 指针指向的位置，还返回至处理器寄存器 `r1` 中（仍须传入有效的 `ok` 指针或 0）。
- 传入的大小参数则返回至寄存器 `r3` 中。
- 根据寄存器 `r1` 的值设置 `N` 标志，故可立即使用 `bpl` 指令进行失败分支。

A3 版本只读存储器启动代码采用如下所示的调用模式，以增强代码抵御硬件攻击的能力，防止指令跳过：

```
// 此时, r0、r2 和 r3 已分别保存 buffer、write 以及 *ok 的参数
1: movs r1, #0 // 将 r1 设为无效的 RCP_VALUE 零值，并将 N 置为 0
2: mov r1, ra // 将 r1 设为大小值（存储于另一个寄存器 a 中）
3: bl validate_ns_buffer // 调用验证函数
4: bpl buffer_invalid // N==0 被视为假（无效）
5: rcp_btrue r1 // 负值应为 0xa500a500
buffer_valid:
6: movs rb, ra // 重新加载大小以便使用
7: rcp_iequal r3, rb // 检查 r3 是否具有正确大小
 // 成功时 r0 保存 'buffer'；该值也可用作检查（如有需要）
8: // 对 `buffer` 和 `size` 执行部分安全代码
 ...
buffer_invalid:
 // r0 保存 BOOTROM_ERROR_INVALID_ADDRESS
 ...
```

#### 5.4.8.30. `xip_setup_func_ptr`

代码: 'X', 'F'

类型: `void *(xip_setup_func_ptr)(void)`

## 5.5. USB 大容量存储接口

只读存储器提供了一个标准 USB 引导程序，使通过 UF2 文件将代码复制到 RP2350 的可写驱动器成为可能（参见第 5.5.2 节）。

将合适的 UF2 文件复制到驱动器后，文件将被下载并写入闪存或 RAM，设备自动重启，使仅通过 USB 连接即可轻松下载和运行 RP2350 代码。

### 5.5.1. RP2350 驱动

RP2350 作为一个标准 128MB 闪存驱动器出现，命名为 `RP2350`，格式为单一 FAT16 分区。驱动器上仅显示两个实际文件。

- `INFO_UF2.TXT` 包含UF2引导程序及版本的字符串描述。
- `INDEX.HTM` 重定向到有关RP2350设备的信息。

RP2350 A2 的默认 `INDEX.HTM` 地址为 <https://raspberrypi.com/device/RP2?version=5A09D5312E22>。对于其他 RP2350 版本，version 参数将有所变化；前六个字符来自芯片的 git 哈希，后六个字符来自 bootrom 的 git 哈希。这些文件的内容及驱动器名称均可进行定制。更多信息请参见第5.7节。

主机可以向 USB 驱动器写入任何类型的文件；然而，通常这些文件并未实际存储，其显示存在仅因主机端缓存所致。

当写入合适的 UF2 文件到设备时，设备会识别特殊内容，并将数据写入 RAM 或闪存的指定位置。

RP2350 上写入闪存的 UF2 文件位置由 UF2 内容的*family id*及分区表决定。

如果不存在分区表，则UF2文件存储于其指定地址；否则（特殊的 *ABSOLUTE*系列ID除外），它们存储于单一分区中，其中UF2闪存地址 `0x10000000`映射至该分区起始位置。

基于分区表或系列ID，UF2文件可能无法下载至闪存的任何位置，此时该文件将被忽略。可通过GET\_INFO - UF2\_STATUS 命令获取更多详细信息。在完整下载一个有效的UF2文件后，RP2350会自动重启以执行新下载的代码。

无效的UF2文件可能完全无法写入，或仅部分写入至RP2350后失败。并非所有操作系统在写入失败后都会通知磁盘写入错误。您可以使用 `picotool verify`命令验证UF2文件是否已正确写入RP2350。

### 5.5.2. UF2格式详情

本节描述了UF2文件下载有效性的必要约束条件。

#### 💡 提示

要生成 UF2 文件，您可以使用位于 `picotool` 中的 `picotool uf2 conver t` 功能。

所有发送至设备的数据必须为符合以下条件的 UF2 块：

- 必须存在一个 `familyID`，其值位于保留范围 `0xe48bff58` 至 `0xe48bff5b` 之间，或为分区表中配置的用户 `family ID`（参见 第 5.5.3 节的表格）。
- 一个大小为 256 的 `payload_size`。

所有数据必须发送到以下内存范围内（依据下载的二进制文件类型，该类型由第一个遇到的 UF2 块的地址决定），且必须完全位于该范围内：

- 常规闪存映像
  - `0x10000000-0x12000000` 闪存：所有块必须以 256 字节对齐为目标地址。超出物理闪存末端的写入将回绕至闪存起始地址。
- 仅限 RAM 映像
  - `0x20000000-0x20082000` 主 RAM：块可按字节对齐定位。
  - `0x13ffc000-0x14000000` XIP RAM：（由于未针对闪存，闪存缓存可用作与 `main RAM` 属性相同的 RAM）

**i 注意**

传统上，UF2仅用于写入闪存，但这主要是因为使用无元数据的.BIN文件作为生成UF2文件源的限制。RP2350充分利用UF2固有的灵活性，支持由构建生成的更丰富的.ELF格式二进制文件作为UF2文件的源。

- numBlocks必须指定适合上述区域的二进制总大小。
- 更改 numBlocks或二进制类型（由 UF2块目标地址确定）将丢弃当前正在进行的传输。
- 更改 familyID将丢弃当前正在进行的传输。
- 所有面向设备的数据必须存在于未标记为UF2\_FLAG\_NOT\_MAIN\_FLASH的块中，该标记表示内容应被忽略，而非区分闪存与RAM。

**i 注意**

当定位到闪存时，UF2块目标地址被解释为位于以 0x1为起始地址的闪存二进制内容中。0000000. UF2映像可能被下载到闪存中其他位置起始的分区，因此实际存储地址为uf2\_image\_target\_base + uf2\_block\_target\_addr - 0x1。0000000.

闪存始终以4 KB扇区为单位擦除，因此在闪存二进制UF2中仅包含扇区内部分256字节页的数据时，扇区内其余的256字节页将保持擦除但内容未定义。

当在单次有效传输过程中，每个 numBlocks块至少被接收一次时，该二进制文件即被视为“已下载”。为防止主机重复发送重复块，块的数据仅在首次接收时写入。

UF2完全下载后，RP2350将重新启动，表面目的是运行新的二进制文件。鉴于RP2350支持向分区下载多种可执行及非可执行UF2，分区中包含一标志，可用于逐案关闭该重启行为。

**i 注意**

在闪存下载后重新启动时，将执行闪存更新引导。因此，在A/B选择中会优先考虑新写入的分区，但如果在较早的分区中发现另一个可引导映像，则不会启动。在RAM下载后重新启动时，映像搜索从下载块的最低地址开始（若主RAM与闪存缓存共存，则主RAM视为较低地址，且搜索范围仅限主RAM或闪存缓存中的一个）。

主机软件可以通过PICOBOOT接口临时禁用UF2写入，以防止与该接口执行的操作相冲突（见下文）；在此情况下，任何正在进行的UF2文件写入将被中止。

**i 注意**

若在下载UF2时发生问题，则表现为设备不会重新启动，即看似未发生任何情况。picotool命令uf2 info可用于查询最后一次下载的状态（另见GET\_INFO - UF2\_STATUS）。

### 5.5.3. UF2目标规则

当下载UF2的第一个块时，系统依据UF2的family ID选择将UF2存储于闪存中的位置。此选择由与get\_uf2\_target\_partition() API（见第5.4.8.18节）相同的代码执行。

以下family ID由bootrom预定义。然而，用户可使用自定义ID以实现更精确的目标定位：

表 455。 RP2350  
bootrom 认可的标  
准 UF2 family ID 表

| 名称                         | 值                       | 描述                                                          |
|----------------------------|-------------------------|-------------------------------------------------------------|
| <code>absolute</code>      | <code>0xe48bff57</code> | 特定 family ID，用于内容直接写入闪存，忽略所有分区。                             |
| <code>rp2040</code>        | <code>0xe48bff56</code> | RP2040 可执行映像。                                               |
| <code>data</code>          | <code>0xe48bff58</code> | 通用数据型 UF2 的通配类别。                                            |
| <code>rp2350_arm_s</code>  | <code>0xe48bff59</code> | RP2350 Arm 安全映像（由 bootrom 负责启动）。                            |
| <code>rp2350_riscv</code>  | <code>0xe48bff5a</code> | RP2350 RISC-V 镜像。                                           |
| <code>rp2350_arm_ns</code> | <code>0xe48bff5b</code> | RP2350 Arm 非安全镜像。不能被只读存储器直接启动。<br>然而，安全用户代码可能需要能够定位此类二进制文件。 |

### 注意

决定存储 UF2 位置的算法唯一可用信息是 UF2 系列 ID；  
该算法无法查看 UF2 内容，因为 UF2 数据扇区可能按任意顺序出现在设备中。

带有 `absolute` 家族 ID 的 UF2 文件下载时不考虑分区边界。分区表（如存在）或 OTP 配置可定义是否允许 **绝对** 系列 ID 下载，并下载至闪存起始地址。默认出厂设置允许 **绝对** 系列 ID 下载。

若存在分区表，其他系列 ID 将下载至单一分区；若不存在分区表，则默认允许 `data`、`rp2350-arm-s`（启用 Arm 架构时）及 `rp2350-riscv`（启用 RISC-V 架构时）系列 ID，并且 UF2 始终下载至闪存起始地址。

如果存在分区表，则最多对分区表进行四次遍历（从遇到的第一个分区开始至最后一个），直到找到匹配的分区；每次遍历的选择标准各不相同：

1. 查找一个（未被占用的）**A** 分区，忽略那些为当前 CPU 架构标记为 `NOT_BOOTABLE` 的分区

使用 `NOT_BOOTABLE` 标志可让您为每种 CPU 架构（Arm 或 RISC-V）设置独立的启动分区；若未在此场景中使用 `NOT_BOOTABLE` 标志，假设首次遇到的分区是 Arm `IMAGE_DEF`，则在启用自动架构切换的 RISC-V 架构下启动时，bootrom 会直接切换回 Arm 架构以便启动 Arm 二进制文件。在分区表中将第一个分区标记为 `NOT_BOOTABLE_RISCV` 可解决此问题。

所谓正确的 CPU 架构，指的是 UF2 的架构（由 `rp2350_arm_s` 或 `rp2350_riscv` 的家族 ID 确定）与当前 CPU 架构相匹配。

此过程允许用户放置 Arm 或 RISC-V UF2 文件，并根据 `NOT_BOOTABLE` 标志的情况，将其存储为所需的格式。

2. 如果启用了自动架构切换且另一架构可用，则查找（未拥有的）**A** 分区，忽略针对该 CPU 架构标记为 `NOT_BOOTABLE` 的分区。

此过程旨在匹配从另一架构启动映像作为后备启动的场景。如果存在因自动架构切换而可能启动的分区，则此处是存储该替代架构 UF2 的合理位置。

3. 查找任何未拥有且接受该系列 ID 的 **A** 分区。

此过程提供根据系列 ID 将任何 UF2 定位至分区的方法，但默认您更倾向于将 UF2 放入匹配的顶级分区，而非已拥有的分区。

4. 最后，查找任何接受该系列 ID 的 **A** 分区。

此过程隐含仅检查已拥有的分区，因为未拥有的分区已在前一过程被匹配。

如果没有任何通道找到匹配，则不会下载 UF2 内容。可以使用 `picotool` 命令 `uf2 info`

来确定最后一次下载的状态（参见GET\_INFO - UF2\_STATUS）。

### 5.5.3.1. A/B 分区及所有权

上述每个通道均指寻找一个 A分区。任何非 B分区均视为 A分区；未配对的分区归类为 A分区。

若找到的 A分区未与 B分区配对，则该 A分区为UF2目标分区。

然而，若 A分区对应有 B分区，则须进一步决定应针对哪个 A/B分区。

1. 若 A分区无所有者，则根据这些分区中任何当前有效的 IMAGE\_DEF 来确定分区选择。没有选择版本号更高的有效分区；在可执行的 IMAGE\_DEF 情况下，情况与启动时相反；这是合理的，因为您希望将UF2文件放到当前未启动的分区。
2. 如果 A分区已标记为已拥有，则假定 A分区和 B分区内不包含可用于基于版本选择的IMAGE\_DEF。因此，使用 A分区（A所有者）及其B分区（B所有者）的所有者信息来进行选择。

然而，是否希望目标为分区 A或分区 B的UF2文件写入A分区，取决于具体使用场景；当A所有者拥有的 IMAGE\_DEF 版本较高（若该 IMAGE\_DEF 可执行则会启动）或当 B所有者拥有的IMAGE\_DEF 版本较高时，选择可能不同。默认情况下，bootrom 会选择分区A，前提是分区A所有者的 IMAGE\_DEF 版本较高，但此行为可通过设置分区 A中的 UF2\_DOWNLOAD\_AB\_ON\_BOOTABLE\_OWNER\_AFFINITY 标志进行更改。

### 5.5.3.2. 多个 UF2 系列

在同一 UF2 文件中包含针对不同系列 ID 的扇区是允许的。UF2 规范旨在允许单一文件支持多种不同设备，但每个设备预期仅接受一个 UF2 系列 ID。

类似地，在 RP2350 上，仅支持下载包含多个系列 ID 的 UF2 文件，条件是根据上述规则，只有其中一个系列 ID 被允许下载至设备。

## 5.6. USB PICOBLOCK 接口

PICOBLOCK 接口是一种用于 RP2350 处于 BOOTSEL 模式时进行交互的低级 USB 协议。此接口可与 USB 大容量存储接口并行使用。

它支持对RAM或闪存的灵活读写、设备重启、在设备上执行代码以及其他若干管理功能。

与接口相关的常量和结构体定义可在SDK头文件 [picoblock.h](#) 中找到。

### 5.6.1. 设备识别

RP2350设备可通过其设备描述符中的供应商ID和产品ID识别（见表456），除非在OTP中设置了不同的值（参见第5.7节）。

表456。RP2350  
引导设备  
描述符

| 字段              | 值  |
|-----------------|----|
| bLength         | 18 |
| bDescriptorType | 1  |

| 字段                 | 值                    |
|--------------------|----------------------|
| bcdUSB             | 2.10                 |
| bDeviceClass       | 0                    |
| bDeviceSubClass    | 0                    |
| bDeviceProtocol    | 0                    |
| bMaxPacketSize0    | 64                   |
| idVendor           | 0x2e8a - 此值可在OTP中覆盖  |
| idProduct          | 0x000f - 此值可在OTP中覆盖  |
| bcdDevice          | 1.00 - 此值可能会在OTP中被覆盖 |
| iManufacturer      | 1                    |
| iProduct           | 2                    |
| iSerial            | 3                    |
| bNumConfigurations | 1                    |

### 5.6.2. 接口识别

PICOBOOT 接口通过厂商特定的接口类、接口子类为零以及接口协议（见表 457）进行识别。

请勿依赖接口编号，该编号取决于设备当前是否暴露大容量存储接口。设备可能根本未暴露 PICOBOOT 接口，因此不可假定其存在。

表 457。PICOBOOT  
接口描述符

| 字段                 | 值           |
|--------------------|-------------|
| bLength            | 9           |
| bDescriptorType    | 4           |
| bInterfaceNumber   | 可变          |
| bAlternateSetting  | 0           |
| bNumEndpoints      | 2           |
| bInterfaceClass    | 0xff (厂商特定) |
| bInterfaceSubClass | 0           |
| bInterfaceProtocol | 0           |
| iInterface         | 0           |

### 5.6.3. 端点识别

PICOBOOT 接口提供一个 **BULK\_OUT** 端点和一个 **BULK\_IN** 端点。这些端点可通过其方向和类型进行识别。禁止依赖端点编号。

## 5.6.4. PICOBLOCK命令

这两个批量端点用于发送命令以及接收成功的命令结果。所有命令均为32字节（详见表458），并发送至 **BULK\_OUT** 端点。

表458 PICOBLOCK  
命令定义

| 偏移量  | 名称              | 描述                                 |
|------|-----------------|------------------------------------|
| 0x00 | dMagic          | 值为 0x431fd10b                      |
| 0x04 | dToken          | 用户提供的用于标识该请求的令牌                    |
| 0x08 | bCmdId          | 命令ID。注意，最高位指示数据传输方向<br>(0x80 = IN) |
| 0x09 | bCmdSize        | args字段中有效数据的字节数                    |
| 0x0a | 保留              | 0x0000                             |
| 0x0c | dTransferLength | 主机期望通过大容量通道发送或接收的字节数               |
| 0x10 | 参数              | 16字节的命令特定数据，零填充                    |

如果发送的命令无效或无法识别，大容量端点将被阻塞。更多信息可通过 **GET\_COMMAND\_STATUS** 请求获取（参见第5.6.5.2节）。

在初始32字节数据包之后，若 **dTransferLength** 非零，则通过大容量通道传输相应字节数，并以反向的空数据包完成命令。若 **dTransferLength** 为零，则以空的 **IN** 数据包表示命令成功。

支持以下命令（为更加清晰，省略了通用字段 **dMagic**、**dToken** 和 **reserved**）

### 5.6.4.1. EXCLUSIVE\_ACCESS (0x01)

声明或释放通过USB对RP2350进行写操作的独占访问权限（相较于大容量存储接口）

表459。PICOBLOCK  
EXCLUSIVE\_ACCESS  
命令结构

| 偏移量  | 名称              | 值 / 描述                  |                                               |
|------|-----------------|-------------------------|-----------------------------------------------|
| 0x08 | bCmdId          | 0x01 (EXCLUSIVE_ACCESS) |                                               |
| 0x09 | bCmdSize        | 0x01                    |                                               |
| 0x0c | dTransferLength | 0x00000000              |                                               |
| 0x10 | bExclusive      | NOT_EXCLUSIVE (0)       | 对USB大容量存储操作无任何限制                              |
|      |                 | EXCLUSIVE (1)           | 禁用USB大容量存储写入（主机应将其视为写保护失败，且任何正在进行的UF2下载均将被中止） |
|      |                 | EXCLUSIVE_AND_EJECT (2) | 通过将驱动介质标记为不存在（弹出驱动器）来锁定USB大容量存储接口             |

### 5.6.4.2. REBOOT (0x02)

RP2350 不支持该功能。

请改用第 5.6.4.10 节。

### 5.6.4.3. FLASH\_ERASE (0x03)

擦除连续范围的闪存扇区。

表 460。PICOBOOT  
FLASH\_ERASE  
命令结构

| 偏移量  | 名称              | 值 / 描述                             |
|------|-----------------|------------------------------------|
| 0x08 | bCmdId          | 0x03 (FLASH_ERASE)                 |
| 0x09 | bCmdSize        | 0x08                               |
| 0x0c | dTransferLength | 0x00000000                         |
| 0x10 | dAddr           | 闪存中要擦除的地址，从该位置开始。该地址必须按照扇区（4 kB）对齐 |
| 0x14 | dSize           | 要擦除的字节数。该数值必须是扇区（4 kB）的整数倍         |

### 5.6.4.4. 读取 (0x84)

从 RP2350 读取连续内存区域（闪存、RAM 或只读存储器）

表 461。PICOBOOT  
读取内存  
命令（闪存、RAM、  
只读存储器）结构

| 偏移量  | 名称              | 值 / 描述                   |
|------|-----------------|--------------------------|
| 0x08 | bCmdId          | 0x84 (READ)              |
| 0x09 | bCmdSize        | 0x08                     |
| 0x0c | dTransferLength | 必须与 dSize 相同             |
| 0x10 | dAddr           | 要读取的地址。可位于闪存、RAM 或只读存储器中 |
| 0x14 | dSize           | 要读取的字节数                  |

### 5.6.4.5. 写入 (0x05)

在 RP2350 上写入连续内存区域（闪存或 RAM）。

表 462。PICOBOOT  
写内存  
命令（闪存、  
RAM）结构

| 偏移量  | 名称              | 值 / 描述                                                  |
|------|-----------------|---------------------------------------------------------|
| 0x08 | bCmdId          | 0x05 (写入)                                               |
| 0x09 | bCmdSize        | 0x08                                                    |
| 0x0c | dTransferLength | 必须与 dSize 相同                                            |
| 0x10 | dAddr           | 写入的起始地址。可以是闪存或RAM，但若为闪存，必须按页（256字节）对齐。闪存必须先擦除，否则结果无法预料。 |
| 0x14 | dSize           | 写入的字节数。若写入闪存且大小非页（256字节）的整倍数，则最后一页将以零填充至页尾。             |

### 5.6.4.6. EXIT\_XIP (0x06)

此为兼容RP2040而设的空操作。进入USB引导程序前，将执行一次XIP退出序列（`flash_exit_xip()`），使外部QSPI设备从任何XIP状态返回至串行命令状态，且外部QSPI设备将保持该状态直至重启。

表463。PICOBOOT  
EXIT\_XIP命令  
结构

| 偏移量  | 名称              | 值 / 描述          |
|------|-----------------|-----------------|
| 0x08 | bCmdId          | 0x06 (EXIT_XIP) |
| 0x09 | bCmdSize        | 0x00            |
| 0x0c | dTransferLength | 0x00000000      |

#### 5.6.4.7. ENTER\_XIP (0x07)

为兼容 RP2040 而提供的空操作指令。请注意，与 RP2040 不同，低级 bootrom 闪存操作不会使 QSPI 接口处于 XIP 无法访问的状态，因此无需每次重新初始化该接口。XIP 设置在进入 USB 引导加载程序之前执行一次，使用固定时钟分频为 6 的 03h 命令。

表 464。PICOBOOT  
进入执行就地 (XIP)  
命令

| 偏移量  | 名称              | 值 / 描述           |
|------|-----------------|------------------|
| 0x08 | bCmdId          | 0x07 (ENTER_XIP) |
| 0x09 | bCmdSize        | 0x00             |
| 0x0c | dTransferLength | 0x00000000       |

#### 5.6.4.8. EXEC (0x08)

RP2350 不支持该功能。

#### 5.6.4.9. VECTORIZE\_FLASH (0x09)

RP2350 不支持该功能。

#### 5.6.4.10. REBOOT2 (0x0a)

使 RP2350 从 BOOTSEL 模式重新启动。请注意，如果未找到有效的可启动映像，可能会重新进入BOOTSEL模式。

参数 *flags*、*delay\_ms*、*p0*、*p1*与api\_reboot()相同

表465。PICOBOOT  
REBOOT2命令  
结构体

| 偏移量  | 名称              | 值 / 描述         |
|------|-----------------|----------------|
| 0x08 | bCmdId          | 0x0a (REBOOT2) |
| 0x09 | bCmdSize        | 0x10           |
| 0x0c | dTransferLength | 0x00000000     |
| 0x10 | dAddr           | 标志             |
| 0x14 | dSize           | delay_ms       |
| 0x18 | dSize           | p0             |
| 0x1c | dSize           | p1             |

#### 5.6.4.11. GET\_INFO (0x8b)

用于从设备检索信息的通用通道。

传输长度指示要检索的最大字节数。返回的第一个字表示随后数据的有效字数。始终返回完整的“传输长度”，必要时以零填充。

下文的“字0”指实际响应的第一个字（计数字之后的字）。

表466。PICOBEST  
GET\_INFO命令  
结构体

| 偏移量  | 名称              | 值 / 描述                        |
|------|-----------------|-------------------------------|
| 0x08 | bCmdId          | 0x0b (GET_INFO)               |
| 0x09 | bCmdSize        | 0x10                          |
| 0x0c | dTransferLength | 待接收数据的大小。注意，此值必须为4的倍数，且小于256。 |

| 偏移量  | 名称    | 值 / 描述                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x10 | bType | <p>正在检索的信息类型：</p> <ul style="list-style-type: none"> <li>• 0x1 - <b>INFO_SYS</b>: 从 get_sys_info() 检索信息；该函数的 <b>flag</b> 参数来自 <b>dParam0</b>。</li> <li>• 0x2 - <b>PARTITION</b>: 从 get_partition_table_info() 检索信息；该函数的 <b>flags_and_partition</b> 参数来自 <b>dParam0</b>。</li> <li>• 0x03 - <b>UF2_TARGET_PARTITION</b>: 检索指定 UF2 family_id 在 BOOTSEL 模式下若被拖入 USB 驱动器时，将被下载至的分区。family id 通过 <b>dParam0</b> 传入。 <ul style="list-style-type: none"> <li>◦ 字 0：目标分区编号： <ul style="list-style-type: none"> <li>■ 0-15 : family 将下载至的分区编号</li> <li>■ 0xff: family 将被下载至绝对位置</li> <li>■ -1 : 无可用位置下载该 family</li> </ul> </li> <li>◦ 字 1：目标分区，第 5.9.4.2 节（当分区编号不为 -1 时） <ul style="list-style-type: none"> <li>◦ 字 2：目标分区，第 5.9.4.2 节（当分区编号不为 -1 时）</li> </ul> </li> </ul> </li> <li>• 0x04 - <b>UF2_STATUS</b>: 获取当前或最近一次 UF2 下载的信息 <ul style="list-style-type: none"> <li>◦ 字 0 - 0nnrrr00af <ul style="list-style-type: none"> <li>■ 'n' - 无重启标志；若为 0x01，则 UF2 下载完成后不进行重启</li> <li>■ 'r' - 若为 0x01，表示正在下载的是 RAM UF2</li> <li>■ 'a' - UF2 下载中止原因标志 <ul style="list-style-type: none"> <li>■ 0x1 EXCLUSIVELY_LOCKED</li> <li>■ 0x2 BAD_ADDRESS</li> <li>■ 0x4 WRITE_ERROR</li> <li>■ 0x8 REBOOT_FAILURE// 若 UF2 目标为禁用架构</li> </ul> </li> <li>■ 'f' - UF2 下载状态标志 <ul style="list-style-type: none"> <li>■ 0x1 IGNORED_FAMILY</li> </ul> </li> </ul> </li> <li>◦ 字 1 - 当前系列 ID</li> <li>◦ 字 2 - 成功下载的 256 字节块数量</li> <li>◦ 字 3 - UF2 中需下载的 256 字节块总数</li> </ul> </li> </ul> |

#### 5.6.4.12. OTP\_READ (0x8c)

从 OTP 中读取数据。（另请参见提供数据的 otp\_access() 函数）返回的数据受“BL”OTP 权限限制，该权限定义引导程序对 OTP 的访问权限。

表 467. PICOBOOT  
OTP\_READ 命令  
结构体

| 偏移量  | 名称              | 值 / 描述                                                                                                                 |
|------|-----------------|------------------------------------------------------------------------------------------------------------------------|
| 0x08 | bCmdId          | 0x8c (OTP_READ)                                                                                                        |
| 0x09 | bCmdSize        | 0x05                                                                                                                   |
| 0x0c | dTransferLength | $wRowCount \times (bEcc ? 2 : 4)$                                                                                      |
| 0x10 | wRow            | 首个读取的行号                                                                                                                |
| 0x12 | wRowCount       | 读取的行数                                                                                                                  |
| 0x14 | bEcc            | <ul style="list-style-type: none"> <li>0 - 读取原始行时（每行返回 32 位数据，最高 8 位为零）</li> <li>1 - 读取 ECC 行时（每行返回 16 位数据）</li> </ul> |

#### 5.6.4.13. OTP\_WRITE (0x0d)

从 OTP 中读取数据。（另请参见执行该操作的 otp\_access() 函数）写入受“BL”OTP权限限制，该权限定义引导加载程序OTP访问权限。

表 468。PICOBOOT  
OTP\_WRITE 命令  
结构体

| 偏移量  | 名称              | 值 / 描述                                                                                                                     |
|------|-----------------|----------------------------------------------------------------------------------------------------------------------------|
| 0x08 | bCmdId          | 0x0d (OTP_WRITE)                                                                                                           |
| 0x09 | bCmdSize        | 0x05                                                                                                                       |
| 0x0c | dTransferLength | $wRowCount \times (bEcc ? 2 : 4)$                                                                                          |
| 0x10 | wRow            | 首个读取的行号                                                                                                                    |
| 0x12 | wRowCount       | 读取的行数                                                                                                                      |
| 0x14 | bEcc            | <ul style="list-style-type: none"> <li>0 - 若写入原始行（每行提供32位，其中最高8位被忽略）</li> <li>1 - 若写入ECC行（每行提供16位，并连同纠错信息写入OTP）</li> </ul> |

### 5.6.5. 控制请求

以下请求通过默认控制管道发送至接口。

#### 5.6.5.1. INTERFACE\_RESET (0x41)

主机发送此控制请求以重置PICOBOOT接口。该命令：

- 清除每个大容量端点上的HALT状态（若已设置）
- 中止任何正在进行的PICOBOOT或大容量存储传输及任何闪存写入操作（此方法为终止卡顿闪存传输的唯一手段）
  - 清除先前的命令结果
  - 移除EXCLUSIVE\_ACCESS，若因独占访问被弹出，则重新挂载大容量存储驱动器。

表 469. PICOBOOT  
重置 PICOBOOT  
接口控制

| bmRequestType | bRequest  | wValue | wIndex | wLength | 数据 |
|---------------|-----------|--------|--------|---------|----|
| 01000001b     | 01000001b | 0000h  | 接口     | 0000h   | 无  |

该命令成功时返回空包。

#### 5.6.5.2. GET\_COMMAND\_STATUS (0x42)

获取最后一条命令的状态（该命令可能仍在执行中）。PICOBOOT协议命令成功完成后，通过大容量端点确认；若操作仍在执行或失败（致使大容量端点阻塞），则可通过此方法查询操作状态。

表 470. PICOBOOT  
获取最后命令  
状态控制

| bmRequestType | bRequest  | wValue | wIndex | wLength | 数据 |
|---------------|-----------|--------|--------|---------|----|
| 11000001b     | 01000010b | 0000h  | 接口     | 0000h   | 无  |

该命令响应以下16字节数据

表471. PICOBOOT  
获取最后命令  
状态控制  
响应

| 偏移量  | 名称     | 描述         |
|------|--------|------------|
| 0x00 | dToken | 命令中指定的用户令牌 |

| 偏移量  | 名称          | 描述                            |                                                           |
|------|-------------|-------------------------------|-----------------------------------------------------------|
| 0x04 | dStatusCode | OK (0)                        | 命令成功完成（或仍在处理中）                                            |
|      |             | UNKNOWN_CMD (1)               | 未识别的命令ID                                                  |
|      |             | INVALID_CMD_LENGTH (2)        | 命令请求长度不正确                                                 |
|      |             | INVALID_TRANSFER_LENGTH (3)   | 根据命令，数据传输长度错误                                             |
|      |             | INVALID_ADDRESS (4)           | 指定的地址对该命令类型无效；即地址与命令预期的类型（闪存或RAM）不匹配                      |
|      |             | BAD_ALIGNMENT (5)             | 指定的地址未按照命令要求正确对齐。                                         |
|      |             | INTERLEAVED_WRITE (6)         | 大容量存储接口 UF2 写操作干扰了当前操作。该命令因状态未知而被中止。如果您拥有独占访问权限，则不会发生此情况。 |
|      |             | REBOOTING (7)                 | 设备正在重启过程中，命令已被忽略。                                         |
|      |             | UNKNOWN_ERROR (8)             | 发生了某些非特定错误。                                               |
|      |             | INVALID_STATE (9)             | 过去发生或未发生某些事件，导致请求当前无法被处理。                                 |
|      |             | NOT_PERMITTED (10)            | 权限违规，例如对只读闪存分区的写入。                                        |
|      |             | INVALID_ARG (11)              | 参数超出支持的取值范围。                                              |
|      |             | BUFFER_TOO_SMALL (12)         | 提供的缓冲区过小，无法容纳结果。                                          |
|      |             | PRECONDITION_NOT_MET (13)     | 操作失败，须先调用另一个只读存储器引导函数。                                    |
|      |             | MODIFIED_DATA (14)            | 缓存数据已被判定与其计算来源的完整数据版本不一致。                                 |
|      |             | INVALID_DATA (15)             | 数据结构验证未通过。                                                |
|      |             | NOT_FOUND (16)                | 尝试访问不存在的对象；或搜索未果。                                         |
|      |             | UNSUPPORTED_MODIFICATION (17) | 基于之前的写入情况，写入操作不可执行，如尝试清除OTP位。                             |
| 0x08 | bCmdId      | 命令标识(ID)                      |                                                           |
| 0x09 | bInProgress | 当命令仍在执行中时返回1                  | 否则为0                                                      |
| 0x0a | 保留          | (6个字节零值)                      |                                                           |

## 5.7. USB白标化

针对基于RP2350的产品，客户可替换通过USB接口暴露的识别信息以进行品牌重塑。此过程称为**白标化**，可通过在RP2350中指定OTP值实现。

1. 通过USB\_WHITE\_LABEL\_ADDR寄存器写入白标数据结构的OTP地址（详见该寄存器说明以了解数据结构内容）。

2. 初始化您希望覆盖的白标数据结构字段，并将其标记为有效。

3. 设置USB\_BOOT\_FLAGS.WHITE\_LABEL\_ADDR\_VALID以标记白标为有效。

以下字段可修改：

### 5.7.1. USB设备描述符

USB设备描述符包含以下16位数值：

- VID (默认值 0x2e8a)
- PID (默认值 0x000f)
- BCD\_DEVICE (默认值 0x0100)
- LANG\_ID (默认值 0x0409)

### 5.7.2. USB设备字符串

- MANUFACTURER (默认“Raspberry Pi”，最大长度30个UTF-16或ASCII字符)
- PRODUCT (默认“RP2350 Boot”，最大长度30个UTF-16或ASCII字符)
- SERIAL\_NUMBER(默认为设备\_id的大写十六进制字符串；OTP的前4行，最大长度30个UTF-16或ASCII字符)

### 5.7.3. USB配置描述符

USB配置描述符虽非严格意义上的白标，但对用户仍有帮助：

- ATTRIBUTES\_MAX\_POWER\_VALUES (默认值0xfa80，表示bMaxPower为0xfa且bmAttributes=0x80)

### 5.7.4. MSD驱动

- VOLUME\_LABEL (默认值“RP2350”，最大长度11个ASCII字符)

### 5.7.5. UF2 INDEX.HTM文件

格式如下：

```
<html>
 <head>
 <meta http-equiv="refresh" content="0;URL='*REDIRECT_URL*' />
 </head>
 <body>正在重定向至 *REDIRECT_NAME*</body><ht
ml>
```

- REDIRECT\_URL (默认值“https://raspberrypi.com/device/RP2?version=5A09D5312E22”，请注意，该12位十六进制数字为SYSINFO\_GITREF\_RP2350的前6位和bootrom gitref的前6位，最大长度127个ASCII字符)

- REDIRECT\_NAME (默认值“raspberrypi.com”，最大长度127个ASCII字符)

### 5.7.6. UF2 INFO\_UF2.TXT 文件

格式如下：

```
UF2 Bootloader v1.0
型号: MODEL
板卡 ID: BOARD_ID
```

- **MODEL** (默认值 "Raspberry Pi RP2350"，最大长度 127 个 ASCII 字符)
- **BOARD\_ID** (默认值 "RP2350"，最大长度127个ASCII字符)

### 5.7.7. SCSI 查询命令

通过 SCSI Inquiry 命令返回：

- **VENDOR** (默认值 "RPI"，最大长度8个ASCII字符)
- **PRODUCT** (默认值 "RP2350"，最大长度16个ASCII字符)
- **VERSION** (默认值 "1"，最大长度4个ASCII字符)

### 5.7.8. 卷标简单示例

较新版本的 `picotool` 可通过 `picotool otp white-label -s <start row> <JSON filename>` 命令从 JSON 文件加载白标数据。以下为将卷标设置为 "SPOON" 的示例 JSON 文件：

```
{
 "volume": {
 "label": "SPOON"
 }
}
```

<start row> 为写入白标结构的 OTP 行数，例如 0x400。

下方列出了可通过 JSON 文件写入的全部白标字段。`manufacturer`、`product` 及 `serial_number` 字段支持 Unicode 字符，若产品名称需包含特殊字符或表情符号，该字段在 OTP 中每个字符将占用两倍空间。

```
{
 "device": {
 "vid": "0x2e8b",
 "pid": "0x000e",
 "bcd": 2.15,
 "lang_id": "0x0c09",
 "manufacturer": "Test's Pis",
 "product": "Test RP2350?",
 "serial_number": "notnecessarilyanumber",
 "max_power": "0x20",
 "attributes": "0xe0"
 },
 "scsi": {
 "vendor": "TestPi",
 "product": "MyPi",
 "version": "v897"
 }
},
```

```

"volume": {
 "label": "TestPi Boot",
 "redirect_url": "https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf",
 "redirect_name": "数据手册", "model
 ":"My Test Pi", "board_
 id": "TPI-RP2350"
}
}

```

### 5.7.9. 卷标深入示例

以下示例演示如何使用 `picotool` 手动将卷标更改为“SPOON”：

- 首先，将白标结构行定义为 `0x400`：

```
$ picotool otp set -e OTP_DATA_USB_WHITE_LABEL_ADDR 0x400
```

- 接着，由于卷标位于 `OTP_DATA_USB_WHITE_LABEL_ADDR` 的索引 `0x8` 处，需写入地址 `0x408`。将卷标字符串的位置定义为相对于 `OTP_DATA_USB_WHITE_LABEL_ADDR` 偏移 `0x30`。在本例中，“SPOON” 包含 5 个字符，因此向 `0x408` 写入 `0x3005`：

```
$ picotool otp set -e 0x408 0x3005
```

- 随后，写入字符“S”和“P”：

```
$ picotool otp set -e 0x430 0x5053
```

- 然后，写入字符“O”和“O”：

```
$ picotool otp set -e 0x431 0x4f4f
```

- 接着，写入字符“N”：

```
$ picotool otp set -e 0x432 0x4e
```

- 最后，启用有效覆盖以使用新值（第 8 位标记 `VOLUME_LABEL` 覆盖为有效，第 22 位标记 `OTP_DATA_USB_WHITE_LABEL_ADDR` 覆盖为有效）：

```
$ picotool otp set -r OTP_DATA_USB_BOOT_FLAGS 0x400100
```

- 为使更改生效，请重启设备：

```
$ picotool reboot -u
```

## 5.8. UART 启动

UART启动是一种用于从简单主机（如另一微控制器）引导无闪存RP2350的最小接口。该接口默认启用于空白设备，从而允许在多设备板上部署RP2350，无需预先加载固件或编程OTP位。

要选择UART启动，需要将QSPI CS<sub>n</sub>拉低（BOOTSEL模式），并将QSPI SD<sub>1</sub>拉高。只读存储器在设备复位释放后不久检查这些信号。UART TX信号出现在QSPI SD<sub>2</sub>，UART RX信号出现在QSPI SD<sub>3</sub>。

UART模式为 8n1：一个起始位，八个数据位，无奇偶校验，一个停止位。每个UART帧内的数据以最低有效位优先顺序发送和接收。波特率固定为1 Mbaud。

### 5.8.1. 波特率及时钟要求

UART启动的名义波特率为1 Mbaud，由标称48 MHz系统时钟频率分频获得。UART启动使用USB PLL产生系统时钟和UART波特率时钟，因此必须提供晶体振荡器或向XIN引脚输入稳定时钟。主机波特率须与RP2350的波特率匹配，误差不超过3%。

默认假定晶体振荡器频率为12 MHz，但通过BOOTSEL\_PLL\_CFG和BOOTSEL\_XOSC\_CFG的OTP设置可覆盖该假设，从而由任意支持的晶体产生标称48 MHz系统时钟。相同OTP配置适用于USB和UART启动。

#### 提示

您可以在无需任何OTP配置的情况下，将时钟输入至XIN端口，速度可略快或略慢，只要您相应调整UART波特率。XIN端口允许的频率范围为7.5至16 MHz，受PLL VCO频率范围限制。

### 5.8.2. UART 启动 Shell 协议

在bootrom采样QSPI CS<sub>n</sub>和 SD<sub>1</sub>信号后，bootrom会有数毫秒延迟，以完成从环形振荡器切换到PLL、擦除SRAM等必要步骤，随后将控制权交给非安全UART引导加载程序。

UART引导加载程序通过打印ASCII启动字符串 RP2350 来表明其已准备就绪。该字符串对应的字节依次为 0x52, 0x50, 0x33, 0x35, 0x30。

在发送任何命令前，必须发送特定的敲击序列以解锁该接口。此措施旨在避免GPIO上的噪声引起瞬态效应，确保主机与设备在初始阶段实现良好同步。序列为：0x56、0xff、0x8b、0xe4。这是RP2040 UF2系列ID，选取为一个广为人知的魔术数字。任何以该四字节序列结尾的字节序列均将被检测。

UART引导shell命令始终为主机至设备方向（RP2350接收），由单个命令字节组成，后可选跟随32字节写入负载。RP2350响应时可选附带32字节读取负载，随后回显命令字节。您应等待命令字节回显后再发送下一个命令。

支持的命令如下：

命令 (ASCII)	命令 (十六进制)	描述
n	0x6e	空操作。不执行任何操作，完成后返回确认。用于恢复丢失同步时对接口的ping操作。 回显命令字节 'n'。
w	0x77	向当前读写指针所在位置写入32字节负载。将地址指针增加32。在所有32字节写入内存后， 回显命令字节 'w'。
r	0x72	从当前读写指针的位置读取32字节的有效载荷。将地址指针增加32。在传输完32字节 读取的有效载荷后，回显命令字节 'r'。

命令 (ASCII)	命令 (十六进制)	描述
c	0x63	清除读写指针。指针重置到SRAM的起始位置 0x20000000，并可从此处开始新的读写序列。回显命令字节 'c'。
x	0x78	执行已写入内存的有效载荷。回显命令字节 'x'，然后重启，传递覆盖整个主SRAM的RAM启动搜索窗口。若在发送此命令前已成功写入有效二进制文件至SRAM，则将执行该文件。

未识别的命令仅回显，无其他效果。未来版本可能会添加更多命令。

### 5.8.3. UART 启动编程流程

1. 重置或关闭RP2350设备电源。
2. 驱动 CSn信号为低电平以选择BOOTSEL， SD1信号为高电平以选择UART。
3. 释放复位信号或为设备上电。
4. 等待启动画面字符串通过QSPI SD2 (TX) 传输。
5. 在QSPISD3 (RX) 上发送敲击序列 0x56、0xff、0x8b和0xe4。
6. 发送 'n' 无操作 (nop) 命令以确保接口已唤醒；若无响应，则重新发送敲击序列。
7. 持续发送 'w' 命令，直至整个写入负载传输完成。
8. (可选) 发送 'c' 清除命令以重置地址指针，随后发送 'r' 读取命令以回读并验证负载。
9. 发送 'x' 执行命令以尝试运行负载。

回显最终 'x' 命令后，UART启动程序无任何反馈。此时设备将重启，尝试对非安全UART引导加载程序加载的数据进行只读存储器映像引导。如果只读存储器映像引导失败，启动只读存储器 (bootrom) 将回落至下一个启动源，继续正常引导流程。保持 CSn拉低且SD1拉高将导致启动只读存储器第二次回落至UART引导，重新发送UART启动画面：这表明启动只读存储器未能识别UART引导二进制文件。

### 5.8.4. 恢复卡死的接口

GPIO上的噪声可能导致UART引导shell停止响应命令，例如其误以为主机正处于写入有效载荷的中途，而主机则认为不是。为重新同步至下一条命令的起始位置：

1. 等待1毫秒以使链路进入静止状态
2. 发送33个 'n' NOP命令 (最长命令的大小)
3. 等待1毫秒并清除接收缓存
4. 发送1个 'n' NOP命令，确认设备以回显NOP进行响应

若接口无法恢复，请重启设备后重试。故障可能由以下原因引起：

- GPIO引脚上的噪声（尤其是在长线路或导线上）
- 波特率匹配不正确
- XOSC XIN上的频率参考不稳定
- 电压等级不匹配（例如RP2350上的QSPI\_IOVDD为1.8 V，而主机IO电压为3.3 V）

## 5.8.5. UART 启动二进制文件要求

UART引导二进制文件为普通RAM二进制文件。必须具备有效的 `IMAGE_DEF`，启动路径才能识别其为可引导二进制文件。对 `IMAGE_DEF` 的搜索范围为整个SRAM，但建议将其放置于接近起始处位置，因为启动只读存储器会线性向前搜索 `IMAGE_DEF` 的起始点。

UART引导二进制文件最大尺寸为主SRAM全部容量：520 KB，即532,480字节。

UART引导仅支持加载至SRAM起始地址，故二进制文件必须链接为运行于地址 `0x2`。`0000000`。不支持稀疏加载。您的程序必须作为单一的平铺二进制映像加载。

所有与RAM映像启动相关的安全要求同样适用于UART启动。如启用安全启动，您的二进制文件必须经过签名。同样，若启用了OTP防回滚版本控制，您的二进制文件回滚版本不得低于OTP中存储的版本号。

## 5.9. 元数据块详细信息

### 5.9.1. 块及块循环

块由固定的32位头部、一个或多个条目、指向下一个块的32位相对偏移量及固定的32位尾部组成。块内所有多字节数值均采用小端格式。块必须从字对齐边界开始，且总大小始终为精确的字数（4字节的倍数）。

块中最后一个条目必须为类型 `PICOBIN_BLOCK_ITEM_LAST`，编码块内条目的总字数。

此32位相对链接构成块的链表。为保证链表有效，该链表最终必须回链至首个块，形成闭合块循环；未闭合循环将导致整个链表被忽略。循环规则用于防止将部分覆盖映像中的孤立块错误地视为有效块。

由于启动路径中RAM的限制，块大小被限制为 `PARTITION_TABLE` 为640字节，`IMAGE_DEF` 为384字节。超过此大小的块将被忽略。

下面展示了包含两个项的简单块格式：

项	字	字节	值
头部	0	4	<code>0xffffded3</code>
项 0	1	1	<code>size_flag:1</code> （0表示1字节大小，1表示2字节大小）， <code>item_type:7</code>
		1	<code>s0 % 256</code>
		1	<code>s0 / 256</code> 当 <code>size_flag==1</code> 时，或者对于绝不会超过256字的块的类型专用数据
		1	类型专用数据
		...	...
项 1	<code>1 + s0</code>	1	<code>size_flag:1</code> （0表示1字节大小，1表示2字节大小）， <code>item_type:7</code>
		1	<code>s1 % 256</code>
		1	<code>s1 / 256</code> 当 <code>size_flag==1</code> 时，或者对于绝不会超过256字的块的类型专用数据
		1	类型专用数据
		...	...

项	字	字节	值
LAST_ITEM	$1 + s0 + s1$	1	<code>0xff</code> ( <code>size_flag == 1, item_type == BLOCK_ITEM_LAST</code> )
		2	<code>s1 + s2</code> (其他项的大小)
		1	<code>0x00</code> (填充)
LINK	$2 + s0 + s1$	4	下一块 HEADER 相对于本块 HEADER 的字节偏移位置。此处构成循环，因此单块循环的该值为0。
FOOTER	$3 + s0 + s1$	4	<code>0xab123579</code>

IMAGE\_DEF 和 PARTITION\_TABLE 块通过其首项为 IMAGE\_DEF 或 PARTITION\_TABLE 项进行识别。

描述块的常量定义可在 SDK 的 picobin.h 文件中查阅。

## 5.9.2. 通用块项

以下项目可能出现在 IMAGE\_DEF 或 PARTITION\_TABLE 块中。

### 5.9.2.1. VERSION 项

二进制的主版本号和次版本号，共32位，外加可选的16位回退版本及OTP行列表，可用于读取以确定该设备允许安装的（温度计编码）最低主回退版本。主版本号和次版本号始终存在，而回滚版本号和OTP行列表一般仅在需要回滚保护时包含。

#### 注意

回滚版本号和OTP行列表仅对 IMAGE\_DEF 有效，对于未加固的 RP2350 则予以忽略。

如OTP行条目数为零，则该块无回滚版本号。

字	字节	值
0	1	<code>0x48</code> ( <code>size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_1BS_VERSION</code> )
	1	$2 + ((\text{num\_otp\_row\_entries} != 0) + \text{num\_row\_entries} + 1) / 2$
	1	<code>0x00</code> (填充)
	1	<code>num_otp_row_entries</code>
1	2	次版本号
	2	主版本号
(2)	(2)	回滚版本号 (当 <code>num_otp_entries != 0</code> 时)
	(2)	首个16位OTP行索引 (当 <code>num_otp_entries != 0</code> 时)
...	...	剩余的16位OTP行索引 (如有必要以零填充以满足字边界对齐)

每个OTP行条目表示一组三个OTP行中的第一个的行号（包含1至4095）。三个OTP行各自被读取为24位原始值，通过按位多数投票组合，然后最高有效 1 位的索引决定版本号。因此，单组三个OTP行可编码0到23（含）的回滚版本，或者当全部24位均被设置时，表示至少为24的不确定版本。每新增一个OTP行索引，表示增加一组三个OTP行，其最大版本号增加24。

不同的OTP行条目在OTP中无需连续，但不得重叠，尽管

bootrom无需对此进行检查（启动签名工具可能会检查）。

### 注意

为了使该条目有效，指定OTP行中可用位数必须严格大于回滚版本号。这表明即使我们不了解后续主版本所使用的全部OTP行，仍能确定设备的最小回滚版本高于该区块所指示的回滚版本。

主版本号和次版本号用于区分两个具有相同主回滚版本的二进制文件中哪个版本较新。例如，用于选择从哪个A/B镜像启动。当未指定主回滚版本时，A/B比较将缺失的主版本视为零，但不会执行回滚检查。

#### 5.9.2.2. HASH\_DEF 项

可选项，包含关于如何进行哈希的信息：

字	字节	值
0	1	<code>0x47</code> ( <code>size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_1BS_HASH_DEF</code> )
	1	<code>0x03</code> ( <code>size_lo</code> )
	1	<code>0x00</code> (填充)
	1	<code>0x01</code> ( <code>PICOBIN_HASH_SHA-256</code> )
1	2	被哈希块的字数（不包括块起始的HEADER字）
	2	<code>0x0000</code> (填充)

`block_words_hashed` 若将此项用于签名，必须包含该项。

最新的 `LOAD_MAP` 项（参见第5.9.3.2节），定义了需要哈希的内容。

#### 5.9.2.3. HASH\_VALUE 项

可选项，包含一个哈希值，该哈希值可由只读存储器用于在未使用签名的情况下验证映像或分区表的哈希。

字	字节	值
0	1	<code>0x09</code> ( <code>size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_HASH_VALUE</code> )
	1	<code>0x01 + n</code> 其中 <code>n</code> 为包含的哈希字数量（1-8）
	2	<code>0x0000</code> (填充)
1	4	哈希值（最低有效32位）
...	...	...
n	4	哈希值（最高有效32位）

### 💡 提示

尽管 SHA-256 哈希包含 8 个字，您仍可选择包括较少的字（至少 1 个字）以节省空间，运行时仅比较包含的字数与完整的 8 字哈希。

该 `HASH_VALUE` 项与最近的 `HASH_DEF` 项（第 5.9.2.2 节）配对，后者定义了被哈希的数据。

#### 5.9.2.4. SIGNATURE 项

可选项，包含加密签名，可由只读存储器用于对映像或分区表的哈希内容进行签名校验。

字	字节	值
0	1	<code>0x4b</code> ( <code>size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_SIGNATURE</code> )
	1	<code>0x21</code> (块大小，单位为字)
	1	<code>0x00</code> (填充)
	1	<code>0x01</code> (PICOBIN_SIGNATURE_SECP256K1)
1	4	公钥（最低有效32位）
...	...	...
16	4	公钥（最高有效32位）
17	4	哈希签名（最低有效32位）
...	...	...
32	4	哈希签名（最高有效32位）

该 `SIGNATURE` 项与最近的 `HASH_DEF` 项（第 5.9.2.2 节）配对，后者定义了所校验签名的哈希值。

### 5.9.3. 镜像定义项

#### 5.9.3.1. IMAGE\_DEF 项

`IMAGE_DEF` 项必须为映像定义中的第一个项：

字	字节	值
0	1	<code>0x42</code> ( <code>size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_1BS_IMAGE_TYPE</code> )
	1	<code>0x01</code> (块大小，单位为字)
	2	<code>image_type_flags</code>

这些标志在 SDK 的 `picobin.h` 中定义，现简要概述如下：

位	字段	数值
0-3	映像类型	<ul style="list-style-type: none"> <li>0 <code>IMAGE_TYPE_INVALID</code></li> <li>1 <code>IMAGE_TYPE_EXE</code>: 映像为可执行文件</li> <li>2 <code>IMAGE_TYPE_DATA</code>: 映像有效，但不用于执行</li> <li>3 保留</li> </ul>

位	字段	数值
剩余位专用于映像类型，目前仅对 EXE 类型定义数值：		
4-5	EXE 安全	0 EXE_SECURITY_UNSPECIFIED 1 EXE_SECURITY_NS：映像在非安全模式下运行 2 EXE_SECURITY_S：映像在安全模式下运行 3 保留
6-7	保留	0
8-10	EXE CPU	0 EXE_CPU_ARM：映像适用于 Arm 架构 1 EXE_CPU_RISCV：映像适用于 RISC-V 架构 2-7 保留
11	保留	0
12-14	EXE 芯片	0 EXE_CHIP_RP2040 1 EXE_CHIP_RP2350 2-7 保留
15	EXE TBYB	0 未设置 1 EXE_TBYB：映像标记为“先试用后购买”

### 5.9.3.2. LOAD\_MAP 条目

具有与 ELF 程序头类似表现形式的可选项。该项既用于定义哈希内容，也用于在映像执行前“加载”数据。例如，安全闪存二进制文件可以在签名校验和执行之前加载到 RAM 中。

加载映射是运行时地址、物理地址、大小及标志的集合。

- 对于“打包”二进制文件，该信息指示只读存储器应在哪里加载代码或数据。
- 对于已哈希或已签名的二进制文件，运行时地址和大小指明必须包含在哈希中以进行验证或签名校验的代码或数据。

#### i 注意

若 runtime\_address 等于 storage\_address，则数据不会被复制，而是在原位进行哈希。

术语说明：

#### 物理地址

数据在映像逻辑地址空间中的存储位置。例如，即使存储于分区中，闪存映像的起始处可能具有物理地址 `0x10000000`。ELF 中最接近的概念是 LMA。

#### 运行时地址

数据在运行时的地址。最接近的 ELF 概念是 VMA。

#### 存储地址

数据存储在闪存中的绝对位置。分区启用时，不一定与闪存的物理地址相同。

RP2350 在 `LOAD_MAP` 中使用物理地址，而非存储地址，因为该数据由处理 ELF 文件的工具编写，该工具不一定知道二进制文件最终存储在闪存中的具体位置。

此操作具备多项用途：

字	字节	值
0	1	<code>0x06 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_LOAD_MAP)</code>
	2	<code>1 + num_entries * 3</code> (块大小, 单位字)
	1	<code>absolute:1, num_entries:7</code>
1-3	加载映射条目 0	
	4	<ul style="list-style-type: none"> <li>如果 <code>absolute == 0</code>  <code>storage_start_address_rel</code> (相对于该 LOAD_MAP 条目地址的存储起始地址)</li> <li>如果 <code>absolute == 1</code>  <code>storage_start_address</code> (绝对存储起始地址)</li> </ul> <p>注意：如果此值为 <code>0x00000000</code>无论 <code>absolute</code>标志的值为何，运行时地址范围均填充为零。在这种情况下，哈希的是32位的 size本身，而非size个零字节。</p>
	4	<code>runtime_start_address</code> (绝对运行时起始地址)
	4	<ul style="list-style-type: none"> <li>如果 <code>absolute == 0</code>  <code>size</code> (内存范围大小, 单位字节)</li> <li>如果 <code>absolute == 1</code>  <code>storage_end_address</code> (绝对存储结束地址)</li> </ul>
	(4-6)	(加载映射条目1)
	...	...

所有地址必须按字对齐，大小必须为4的倍数。在RP2350 A3及更早版本中，bootrom在某些情况下允许大小非4的倍数，但可能无法正常工作。

#### 5.9.3.2.1. 通过LOAD\_MAP实现XIP固定

通常，进入二进制文件时，XIP缓存会被解固定并刷新。这对于加载闪存二进制文件及安全目的均合理。

然而，如果您拥有一个非闪存二进制文件，其代码或数据位于XIP RAM地址空间，则需添加特殊的 LOAD\_MAP 条目用于向只读存储器指示应固定XIP内容。

任何load-map条目（其中`storage_address`等于 `runtime_address`）且大小有效且大于零均适用，例如以下简单的load map：

字	字节	值
0	1	<code>0x06 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_LOAD_MAP)</code>
	2	<code>0x04</code> (以字为单位的块大小)
	1	<code>0x81 (absolute == 1, num_entries == 1)</code>
1-3	加载映射条目 0	
	4	<code>XIP_SRAM_BASE (storage_start_address)</code>
	4	<code>XIP_SRAM_BASE (runtime_start_address)</code>
	4	<code>0x04</code> (以字节为单位的大小)

### 5.9.3.3. VECTOR\_TABLE 项

仅适用于Arm的可选条目，用于指定初始Arm向量表的位置。如果存在，entry\_point/initial\_sp将从此处获取（除非同时存在ENTRY\_POINT条目）。若不存在ENTRY\_POINT或VECTOR\_TABLE条目，则假定Arm向量表位于映像起始处。

字	字节	值
0	1	0x03 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_1BS_VECTOR_TABLE)
	1	0x02 (块大小，单位为字)
	2	0x0000 (填充)
1	4	向量表（运行时）地址

#### 注意

在RISC-V架构上，VECTOR\_TABLE项被忽略。

### 5.9.3.4. ENTRY\_POINT 项

包含初始PC与SP信息的可选项，SP限制地址为可选

字	字节	值
0	1	0x44 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_1BS_ENTRY_POINT)
	1	0x03 或 0x04 (块大小，单位为字)
	2	0x0000 (填充)
1	4	初始PC（运行时）地址（即入口点）
2	4	初始SP地址（即栈指针）
(3)	4	可选的SP限制地址（即栈限制）

### 5.9.3.5. ROLLING\_WINDOW\_DELTA 项

允许二进制文件非在 0x1 地址运行的可选项 0000000。注意，该偏移量是相对于二进制文件存储在闪存分区中所引起的滚动偏移之外的增量。

字	字节	值
0	1	0x05 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_1BS_ROLLING_WINDOW_DELTA)
	1	0x02 (块大小，单位为字)
	2	0x0000 (填充)
1	4	有符号32位增量

该增量表示镜像中的字节偏移量，值为 0x1000000 应被映射。

如果为正数，增量必须是4 KB的整数倍，且允许在二进制起始位置之前“跳过”其他数据。如果为负数，增量必须是4 M B的整数倍，允许运行链接到 0x10400000、0x01080000 和 0x010c0000 以及标准 0x1 地址的闪存二进制文件。0000000

**① 注意**

对于非闪存二进制文件，将忽略`ROLLING_WINDOW_DELTA`条目。

### 5.9.4. 分区表项

分区表允许将32 MB闪存区域（ $2 \times 16\text{ MB}$ ）划分为多个分区。每个分区可以指定权限及其他分区属性，同时也可为未分区空间设定权限。

权限用于指定安全代码、非安全代码及“NSBoot”（即引导加载程序及PICOBOOT）的读写访问权限。

**① 注意**

这些权限仅作为对Secure代码的建议，但`flash_op()`、PICOBOOT闪存访问命令以及UF2下载均予以遵守。

#### 5.9.4.1. PARTITION\_TABLE 项

字	字节	值
0	1	<code>0x44 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_PARTITION_TABLE)</code>
	2	块大小（单位：字）
	1	<code>singleton_flag:1, pad:3, partition_count:4</code>
1	4	未分区空间权限及标志
分区 0		
2	4	分区 0 的权限与位置
3	4	分区 0 的权限及标志
如果 <code>_partition_0_has_id</code> :		
3	4	<code>partition_0_ID_lo</code>
4	4	<code>partition_0_ID_hi</code>
每个额外的 <code>family ID</code> 占用一个字（可无）:		
x	4	<code>partition_0_family ID_0</code>
x + 1	4	<code>partition_0_family ID_1</code>
...		
如果 <code>_partition_0_has_name</code> :		
y	1	<code>reserved:1 (0), name_len_bytes:7</code>
	1	<code>partition_0_name_byte_0</code>
	1	<code>partition_0_name_byte_1</code>
	1	<code>partition_0_name_byte_2</code>

字	字节	值
y+1	1	<i>partition_0_name_byte_3</i>
	1	<i>partition_0_name_byte_4</i>
	1	<i>partition_0_name_byte_5</i>
	1	<i>partition_0_name_byte_6</i>
	...	...
	?	<i>partition_0_name_byte_n_minus_x</i> 至 <i>partition_0_name_byte_n_minus_2</i>
	1	<i>partition_0_name_byte_n_minus_1</i>
(分区 1)		
...	...	...

#### 5.9.4.2. 分区位置、权限及标志

两个公共词存储于分区表中，分别对应未分区空间与各分区。这些公共词描述大小/位置，以及访问权限和各类标志。

权限字段在两个词中均重复出现，因此这两个词分别是permissions\_and\_location和 permissions\_and\_flags。

表472. 权限字段。  
'P'表示该字段适用于分区，'  
U'表示该字段适用于未分区空间，但描述中始终使用“partition”一词。

掩码	适用对象	描述
0x04000000u	'P' 'U'	PERMISSION_S_R_BITS
		若设置，本分区允许安全代码读取。详见第5.1.3节。
0x08000000u	'P' 'U'	PERMISSION_S_W_BITS
		若设置，本分区允许安全代码写入。详见第5.1.3节。
0x10000000u	'P' 'U'	PERMISSION_NS_R_BITS
		若设置，本分区允许非安全代码读取。详见第5.1.3节。
0x20000000u	'P' 'U'	PERMISSION_NS_W_BITS
		若设置，本分区允许非安全代码写入。详见第5.1.3节。
0x40000000u	'P' 'U'	PERMISSION_NSBOOT_R_BITS
		若设置，则该分区可被NSBOOT（引导加载程序）安全代码读取。详见第5.1.3节
0x80000000u	'P' 'U'	PERMISSION_NSBOOT_W_BITS
		若设置，则该分区可被NSBOOT（引导加载程序）安全代码写入。详见第5.1.3节

表473. 位置字段。  
'P'表示该字段适用于分区，'  
U'表示该字段适用于未分区空间，但描述中始终使用“分区”一词

掩码	适用对象	描述
0x00001ffffu	'P' 'U'	LOCATION_FIRST_SECTOR_BITS
		分区内第一个扇区的扇区号（0-4095）（每个扇区大小为4 kB）
0x03ffe000u	'P' 'U'	LOCATION_LAST_SECTOR_BITS
		分区内最后一个扇区的扇区号（0-4095）（每个扇区大小为4 kB）

表474. 标志字段。'P'表示该字段适用于分区，'U'表示该字段适用于未分区空间，但描述中始终使用“分区”一词

掩码	适用对象	描述
0x00000001u	'P'	<b>FLAGS_HAS_ID_BITS</b> 若设置，则该分区具有64位标识符。
0x00000006u	'P'	<b>FLAGS_LINK_TYPE_BITS</b> 存储在分区中的链接类型： <ul style="list-style-type: none"> <li>• 0x0 - 无</li> <li>• 0x1-A_PARTITION：这是一个“B”分区，<a href="#">LINK_VALUE</a>字段存储对应“A”分区的分区号。</li> <li>• 0x2-OWNER：这是一个“A”分区，<a href="#">LINK_VALUE</a>字段存储所属分区的分区号（该所属分区也应为“A”分区）。</li> </ul>
0x00000078u	'P'	<b>FLAGS_LINK_VALUE_BITS</b> 若 <a href="#">LINK_TYPE</a> 非零，则该字段保存链接分区的分区号。
0x00000180u	'P'	<b>FLAGS_ACCEPTS_NUM_EXTRA_FAMILIES_BITS</b> 0-3 表示分区接受的额外非标准UF2系列ID数量。
0x00000200u	'P'	<b>FLAGS_NOT_BOOTABLE_ARM_BITS</b> 若设置，则该分区在Arm架构上标记为不可启动，并将在Arm启动过程中被忽略。 为非Arm可启动分区设置此项可提升启动性能。
0x00000400u	'P'	<b>FLAGS_NOT_BOOTABLE_RISCV_BITS</b> 如果设置，此分区将在RISC-V上标记为不可启动，并在RISC-V启动时被忽略。对非RISC-V启动分区设置此项可提升启动性能。
0x00000800u	'P'	<b>FLAGS_UF2_DOWNLOAD_AB_NON_BOOTABLE_OWNER_AFFINITY</b>
0x00001000u	'P'	<b>FLAGS_HAS_NAME_BITS</b> 如果设置，分区将具有名称。
0x00002000u	'P' 'U'	<b>FLAGS_UF2_DOWNLOAD_NO_REBOOT_BITS</b> 如果设置，RP2350在将UF2文件拖入此分区后不会重新启动。
0x00004000u	'P' 'U'	<b>FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2040_BITS</b> 如果设置，具有 RP2040 系列 ID 0xe48bff56 的 UF2 可下载到该分区。
0x00008000u	'U'	<b>FLAGS_ACCEPTS_DEFAULT_FAMILY_ABSOLUTE_BITS</b> 如果对未分区空间设置此项，具有 ABSOLUTE 系列 ID 0xe48bff57 的 UF2 可以下载到 RP2350，并会按照 UF2 中指定的地址写入，而不考虑分区位置。分区定义的闪存访问权限依然被严格遵守（若需写入只读闪存区域则 UF2 下载将会失败）。
0x00010000u	'P' 'U'	<b>FLAGS_ACCEPTS_DEFAULT_FAMILY_DATA_BITS</b> 若设置，带有 DATA 系列 ID 0xe48bff58 的 UF2 可下载至此分区。
0x00020000u	'P' 'U'	<b>FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2350_ARM_S_BITS</b> 若设置，带有 RP2350_ARM_S 系列 ID 0xe48bff59 的 UF2 可下载至此分区。
0x00040000u	'P' 'U'	<b>FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2350_RISCV_BITS</b> 若设置，带有 RP2350_RISCV_V 系列 ID 0xe48bff5a 的 UF2 可下载至此分区。

掩码	适用对象	描述
0x00080000u	'P' 'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2350_ARM_NS_BITS
		若设置，带有RP2350_ARM_NS系列ID 0xe48bff5b的UF2可下载至此分区。
0x03f00000u	'P' 'U'	保留；应设为0

### 5.9.5. 最小可用镜像元数据

任何二进制文件中必须嵌入最少量元数据（有效的 `IMAGE_DEF` 块），以便启动只读存储器能够识别其为有效程序映像，而非空白闪存内容或断开连接的闪存设备。该元数据必须出现在闪存映像的前4 KB内，或位于RAM或OTP映像的任意位置。

与RP2040不同，闪存二进制文件无需在闪存地址0处包含带校验和的“boot2”闪存初始化函数。RP2350启动只读存储器在扫描闪存时执行简单的尽力而为XIP设置，闪存驻留程序可在该状态下继续执行，或选择稍后重新配置QSPI接口以获得最佳性能。

#### 5.9.5.1 最小Arm `IMAGE_DEF`

假设CRIT1.SECURE\_BOOT\_ENABLE为清除状态，最小有效的 `IMAGE_DEF` 为以下20字节序列：

字	LE 值	字节	描述
0	0xffffded3	4	<code>PICOBIN_BLOCK_MARKER_START</code>
1	0x10210142	1	<code>0x42(item_type == PICOBIN_BLOCK_ITEM_1BS_IMAGE_TYPE)</code>
		1	<code>0x01</code> (项目大小为 1 字)
		2	<code>0x1021</code> <code>(PICOBIN_IMAGE_TYPE_IMAGE_TYPE_AS_BITS(EXE))</code> <code>PICOBIN_IMAGE_TYPE_EXE_SECURITY_AS_BITS(S))</code> <code>PICOBIN_IMAGE_TYPE_EXE_CPU_AS_BITS(Arm))</code> <code>PICOBIN_IMAGE_TYPE_EXE_CHIP_AS_BITS(RP23500))</code>
		2	<code>0xff(size_type == 1, item_type_ == PICOBIN_BLOCK_ITEM_2BS_LAST)</code>
2	0x000001ff	2	<code>0x0001</code> (size)
		1	<code>0x00</code> (pad)
		3	<code>0x00000000</code>
4	0xab123579	4	<code>PICOBIN_BLOCK_MARKER_END</code>

“LE 值”栏表示一个32位小端格式的数值，应准确无误地出现在您的程序镜像中。

由于上述区块未指定明确入口点，bootrom将假定二进制文件以Cortex-M向量表开始，并通过表中指定的复位处理程序及初始堆栈指针（表中+4和+0字节偏移）进入。可以通过`PICOBIN_BLOCK_ITEM_1BS_VECTOR_TABLE`项提供显式向量表指针，或通过`PICOBIN_BLOCK_ITEM_1BS_ENTRY_POINT`项直接指定入口点。

#### 5.9.5.2. 最小RISC-V `IMAGE_DEF`

最小有效的 `IMAGE_DEF` 为以下 20 字节序列：

字	LE 值	字节	描述
0	0xffffded3	4	PICOBIN_BLOCK_MARKER_START
1	0x11010142	1	0x42(item_type == PICOBIN_BLOCK_ITEM_1BS_IMAGE_TYPE)
		1	0x01(项目大小为1字)
		2	0x1101 (PICOBIN_IMAGE_TYPE_IMAGE_TYPE_AS_BITS(EXE) PICOBIN_IMAGE_TYPE_EXE_CPU_AS_BITS(RISCV) PICOBIN_IMAGE_TYPE_EXE_CHIP_AS_BITS(RP23500))
2	0x0000001ff	1	0xff(size_type == 1, item_type == PICOBIN_BLOCK_ITEM_2BS_LAST)
		2	0x0001(size)
		1	0x00(pad)
3	0x00000000	4	指向块循环中下一个块的相对指针 - 0x00000000 表示链接至自身（仅包含该块的循环）
4	0xab123579	4	PICOBIN_BLOCK_MARKER_END

“LE 值”栏表示一个32位小端格式的数值，应准确无误地出现在您的程序镜像中。

由于上述区块未指定明确入口点，bootrom 将从其最低地址进入二进制文件，此为 RISC-V 的默认行为。该默认入口点可通过 `PICOBIN_BLOCK_ITEM_1BS_ENTRY_POINT` 项覆盖。请注意，`PICOBIN_BLOCK_ITEM_1BS_VECTOR_TABLE` 在 RISC-V 上无效，因为与 Cortex-M 不同，RISC-V 的向量表不定义程序入口点。

## 5.10. 启动示例场景

本节描述各种不同启动场景的设置与配置步骤。

### 5.10.1. 安全启动

要在 RP2350 上启用安全启动，您必须：

1. 设置从 `BOOTKEY0_0` 开始使用的启动密钥的 SHA-256 哈希值。
2. 在 `BOOT_FLAGS1.KEY_VALID` 中设置您将使用的密钥对应位。
3. 可选地，在 `BOOT_FLAGS1.KEY_INVALID` 中设置所有未使用密钥对应位——建议这样做以防止恶意行为者日后安装自己的启动密钥。
4. 设置 `CRIT1.SECURE_BOOT_ENABLE` 以启用安全启动。

#### 注意

以上步骤为启动只读存储器中启用安全启动支持的最低要求。有关完全保障设备安全所需采取的额外步骤，例如禁用硬件调试，请参见第10.5节。

以上所有操作均可通过 `picotool` 完成。例如，使用 `picotool seal` 进行签名时，可以添加一个 OTP JSON 输出文件，工具会在该文件中添加相关的 OTP 字段值，以启用安全启动 (`BOOTKEY0_0`、`BOOT_FLAGS1.KEY_VALID` 和 `CRIT1.SECURE_BOOT_ENABLE`)：

```
$ picotool seal --sign unsigned.elf signed.elf private.pem /path/to/otp.json
```

要配置 SDK 在签名时输出该 OTP JSON 文件，请在您的[CMakeLists.txt](#)中添加如下命令：

```
pico_set_otp_key_output_file(target_name /path/to/otp.json)
```

然后，您可以执行以下命令将该 OTP JSON 文件写入设备，从而启用安全启动：

```
$ picotool otp load /path/to/otp.json
```

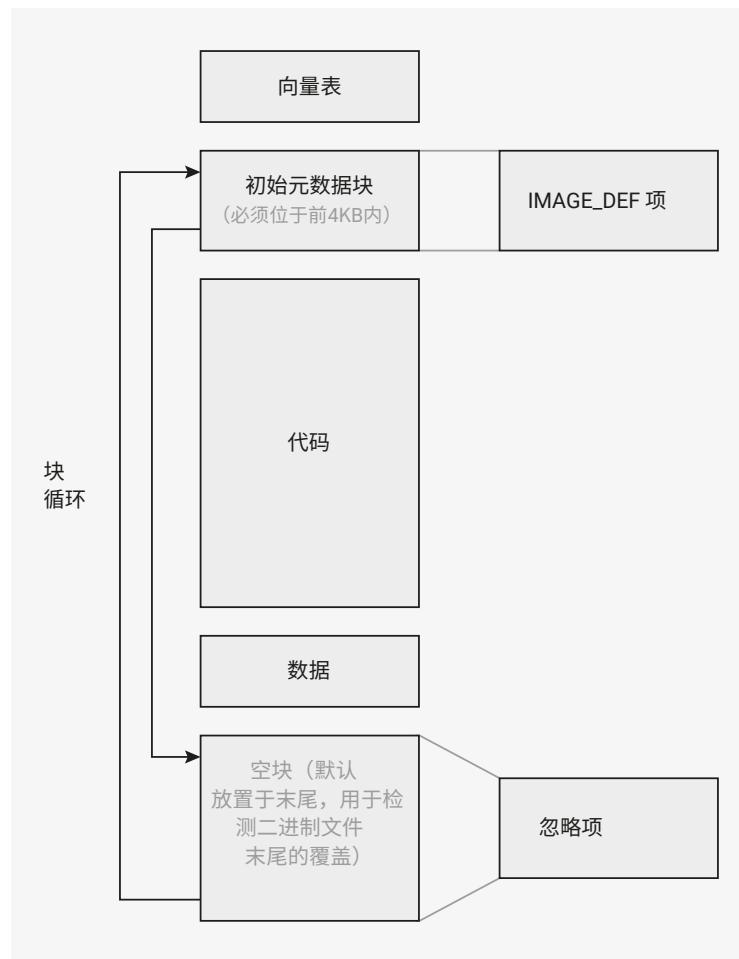
启用安全启动后，bootrom 将验证所有支持介质中镜像的签名：闪存、OTP 以及通过 UART 和 USB 启动加载器预加载至 SRAM 的镜像。此时，您将无法运行未签名的镜像；在开发过程中，您可能会发现禁用安全启动更为方便。下一节介绍了在启用安全启动的设备上运行的**signed images**的生成。

### 5.10.2. 签名映像

#### 💡 提示

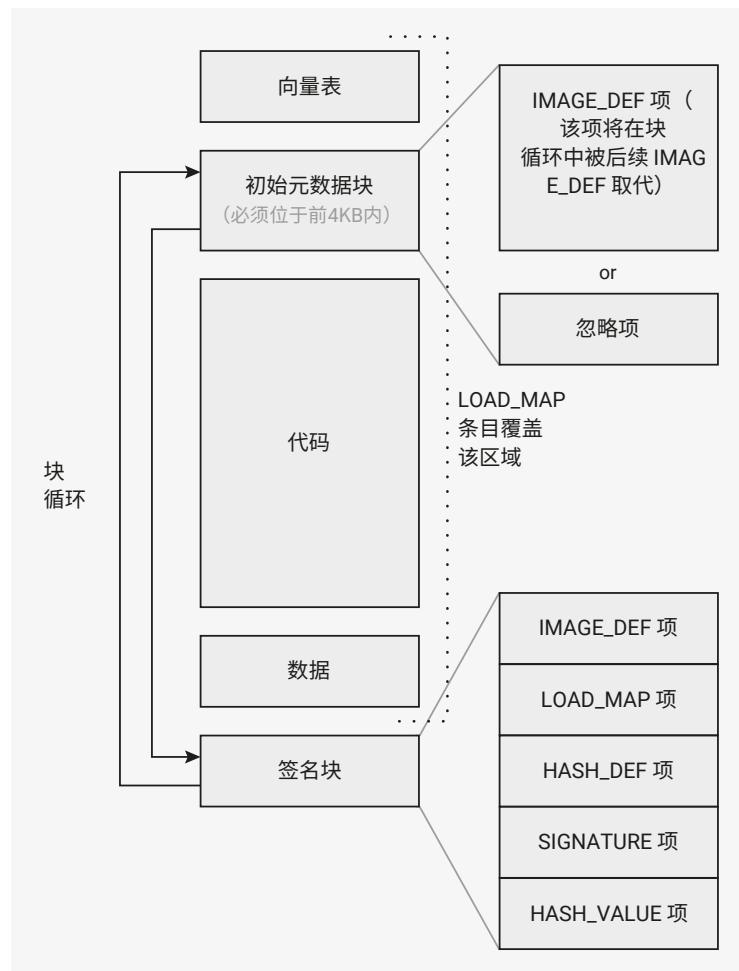
本节涉及第5.1节中描述的块循环和映像定义（及相关的 [IMAGE\\_DEF](#) 数据结构）概念。您应在阅读本节之前先阅读bootrom概念部分。

以下展示了由SDK生成的一个映像（及其块循环）示例。



第一个块必须位于映像的前4KB内，且是描述该映像的 IMAGE\_DEF 块。该块链接到映像末尾的一个空块，空块包含在块循环中，用于帮助检测部分写入的二进制文件。如果映像末尾缺失或被覆盖，则块循环无法正确闭合，视为无效。

`picotool` 可用于对二进制文件进行签名，在此情况下会按照以下方式修改映像：



请注意，镜像末尾的标记块已被新的 **IMAGE\_DEF** 块替换，该块包含首块的信息及附加的新信息。新信息包括签名（或仅进行散列时的散列值），以及一条 **LOAD\_MAP** 条目，指明被签名或散列的镜像区域。

运行时，bootrom 将选择块循环中最后一个有效的 **IMAGE\_DEF** 作为启动项。

签名要求对 **LOAD\_MAP** 中指定的数据进行 **SHA-256** 散列，并对 **HASH\_VALUE** 项指定块的字（必须包括 **SIGNATURE** 项的首字）进行散列。此散列随后使用 **ECDSA secp256k1** 私钥签名，生成存储于 **SIGNATURE** 项中的64字节签名。

为实现安全启动，建议使用封装后的SRAM二进制文件代替闪存二进制文件，因签名校验只在启动阶段执行，若攻击者获得物理访问权限，可能在签名校验后替换外部闪存中的数据，从而运行未签名代码。

在SDK中对二进制文件进行签名和/或哈希处理时，您可将以下函数添加至CMakeLists.txt文件：

```
pico_sign_binary(target_name /path/to/keyfile.pem)
pico_hash_binary(target_name)
```

调用 **pico\_add\_extra\_outputs** 时，此操作将触发 **picotool seal** 命令，对二进制文件执行签名和/或哈希处理。也可通过手动执行以下命令调用 **picotool seal**，对二进制文件进行签名和/或哈希处理：

```
$ picotool seal --sign --hash unsigned.elf signed.elf private.pem
```

### 5.10.3. 打包二进制文件

封装二进制文件为仅限SRAM/XIP RAM使用的二进制文件，已进行后期处理以便存储于闪存中。创建封装SRAM二进制文件时，可将已编译可在SRAM中运行的二进制文件（SDK中的`no_flash`二进制文件）添加相对`LOAD_MAP`项于`IMAGE_DEF`块中，指定SRAM的运行时地址。生成的二进制文件可正常在RAM中运行，或存储于闪存，由启动只读存储器加载至RAM。此`LOAD_MAP`项将在使用`picotool seal`时添加至所有二进制文件。

要在SDK中封装二进制文件，请将以下内容添加到您的CMakeLists.txt文件中。这将使UF2文件定位于闪存起始位置，拖放时自动调用`picotool seal`以添加相应的`LOAD_MAP`。

```
pico_package_uf2_output(target_name 0x10000000)
```

或者，您可以使用绝对的`LOAD_MAP`，其中`storage_address`位于闪存，`runtime_address`位于SRAM。但这些二进制文件仅能在写入闪存后运行，无法直接在SRAM中启动以进行调试。

例如，假设您有一个编译为运行在`0x20000000`且长度为`0x8000`的二进制文件，文件末尾包含一个元数据块，作为第二项（紧随`IMAGE_DEF`之后，即`LOAD_MAP`在该块内偏移8字节）的`LOAD_MAP`，则相对的`LOAD_MAP`内容为：

字	字节	值
0	1	<code>0x06 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_LOAD_MAP)</code>
	2	<code>0x04</code> （以字为单位的块大小）
	1	<code>0x01 (absolute == 0, num_entries == 1)</code>
1-3	加载映射条目 0	
	4	<code>-0x8008 = 0xFFFF7FF8 (storage_start_address_rel)</code>
	4	<code>0x20000000 (runtime_start_address)</code>
	4	<code>0x8000</code> （以字节为单位的大小）

绝对`LOAD_MAP`为：

字	字节	值
0	1	<code>0x06 (size_flag == 0, item_type == PICOBIN_BLOCK_ITEM_LOAD_MAP)</code>
	2	<code>0x04</code> （以字为单位的块大小）
	1	<code>0x81 (absolute == 1, num_entries == 1)</code>
1-3	加载映射条目 0	
	4	<code>0x10000000 (storage_start_address)</code>
	4	<code>0x20000000 (runtime_start_address)</code>
	4	<code>0x2008000 (runtime_end_address)</code>

### 5.10.4. A/B 启动

这是常见的启动场景，允许在不覆盖软件的情况下进行软件更新。一个简单的分区布局如下：

分区 0  
支持的UF2系列：rp2350-arm-s, rp2350-riscv  
分区 1  
支持的UF2系列：rp2350-arm-s, rp2350-riscv  
链接类型：“A”  
链接值：0

这是一个包含两个分区的分区表，其中分区0是分区1的A分区（因此分区1是B分区）。

### *i* 注意

为避免混淆，建议采用最佳实践，使两个分区权限一致且均支持相同的UF2系列。bootrom 仅会检查 A 分区中的 UF2 系列，以确定某个 A/B 组合是否接受特定系列，即使分区 B 不可写，也不会允许向分区 A 下载。

将 UF2 文件拖拽到设备时，系统将定位当前未启动的分区。然后，bootrom 将执行FLASH\_UPDATE操作，启动进入新的二进制文件（详见第 5.1.16 节）。

### *i* 注意

在空白的 A/B 分区状态下，首次下载实际写入分区 B。

使用picotool partition create创建上述分区表时，可采用以下 json：

```
{
 "version": [1, 0],
 "unpartitioned": {
 "families": ["absolute"],
 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 }
 },
 "partitions": [
 {
 "name": "Example A",
 "id": 0,
 "size": "2044K",
 "families": ["rp2350-arm-s", "rp2350-riscv"],
 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 }
 },
 {
 "name": "示例 B",
 "id": 1,
 "size": "2044K",
 "families": ["rp2350-arm-s", "rp2350-riscv"],
 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 },
 "link": ["a", 0]
 }
]
}
```

```

 }
]
}

```

然后可以使用`picotool load`将其安装到设备上，或者如果您将分区表导出为 UF2 文件，则可以通过 UF2 拖放方式安装。

### 5.10.5. 具所有权分区的 A/B 启动

拥有分区的概念适用于以下情形：

- 您需要单独的数据分区（通常不包含块循环），但
- 您希望将这些分区与 A/B 配对中的特定引导分区关联

此场景下的示例分区表如下：

```

分区 0
接受系列: rp2350-arm-s, rp2350-riscv, 分区1

接受系列: rp2350-arm-s, rp2350-riscv
链接类型: "A"
链接值: 0 分区2
接受类别: da
ta 链接类型: "Owner" 链
接值: 0

ignored_during_arm_boot: true
ignored_during_riscv_boot: true

分区3
接受类别: data
链接类型: "A"
链接值: 2
ignored_during_arm_boot: true
ignored_during_riscv_boot: true

```

这是一个包含4个分区的分区表。如前所述，分区0是分区1的A分区（因此分区1为B分区）。此外，分区2是分区3的A分区（因此分区3为B分区）。最后，分区0是分区2的“所有者”分区。

因此，分区2和3“属于”分区0和1。

当向被拥有的分区下载UF2文件时，引导加载程序会根据当前启动的分区0或1，选择目标分区2或3。例如，若当前启动的为分区1（因其版本高于分区0），则所有带有数据家族ID的UF2下载将定位至分区3。

#### 提示

每个分区中均设有标志，可用于切换“affinity”，例如，在上述情形中使数据家族ID定位至分区2而非分区3。

### ⓘ 注意

仅有get\_uf2\_target\_partition() bootrom函数会考虑所有者分区。pick\_ab\_partition()函数始终仅基于传入的A/B分区进行选择，换言之，若传入分区2，则不会查看分区0与1。

使用picotool partition create创建此分区表时，可采用以下JSON：

```
{
 "version": [1, 0],
 "unpartitioned": {
 "families": ["absolute"],
 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 }
 },
 "partitions": [
 {
 "name": "示例A",
 "id": 0,
 "size": 128k,
 "families": ["rp2350-arm-s", "rp2350-riscv"],
 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 }
 },
 {
 "name": "示例B",
 "id": 1,
 "size": 128k,
 "families": ["rp2350-arm-s", "rp2350-riscv"],
 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 },
 "link": ["a", 0]
 },
 {
 "name": "示例a", "id": 2, "size": 20k, "families": ["data"], "permissions": {
 "secure": "rw", "nonsecure": "rw", "bootloader": "rw"
 },
 "link": ["owner", 0],
 "ignored_during_arm_boot": true,
 "ignored_during_riscv_boot": true
 },
 {
 "name": "示例b", "id": 3, "size": 20k, "families": ["data"],
```

```

 "permissions": {
 "secure": "rw",
 "nonsecure": "rw",
 "bootloader": "rw"
 },
 "link": ["a", 2],
 "ignored_during_arm_boot": true,
 "ignored_during_riscv_boot": true
}
]
}

```

然后可以使用`picotool load`将其安装到设备上，或者如果您将分区表导出为 UF2 文件，则可以通过 UF2 拖放方式安装。

### 5.10.6. 自定义引导加载程序

在此情形下，bootloader 会在启动镜像之前运行。该过程可能执行额外的验证，或配置供镜像使用的外设。为了使其生效，块循环必须同时包含引导加载程序的 `IMAGE_DEF` 以及定义闪存布局的 `PARTITION_TABLE`。

在本例中，我们希望引导加载程序存在 A/B 两个版本，因此使用槽 0 和槽 1。有关详细信息，请参见第 5.1.15 节，您可能需要增大槽 0 的大小以容纳引导加载程序。

 **警告**

槽大小的更改不可逆，因此若您在实际操作中尝试，可选择省略槽 1。

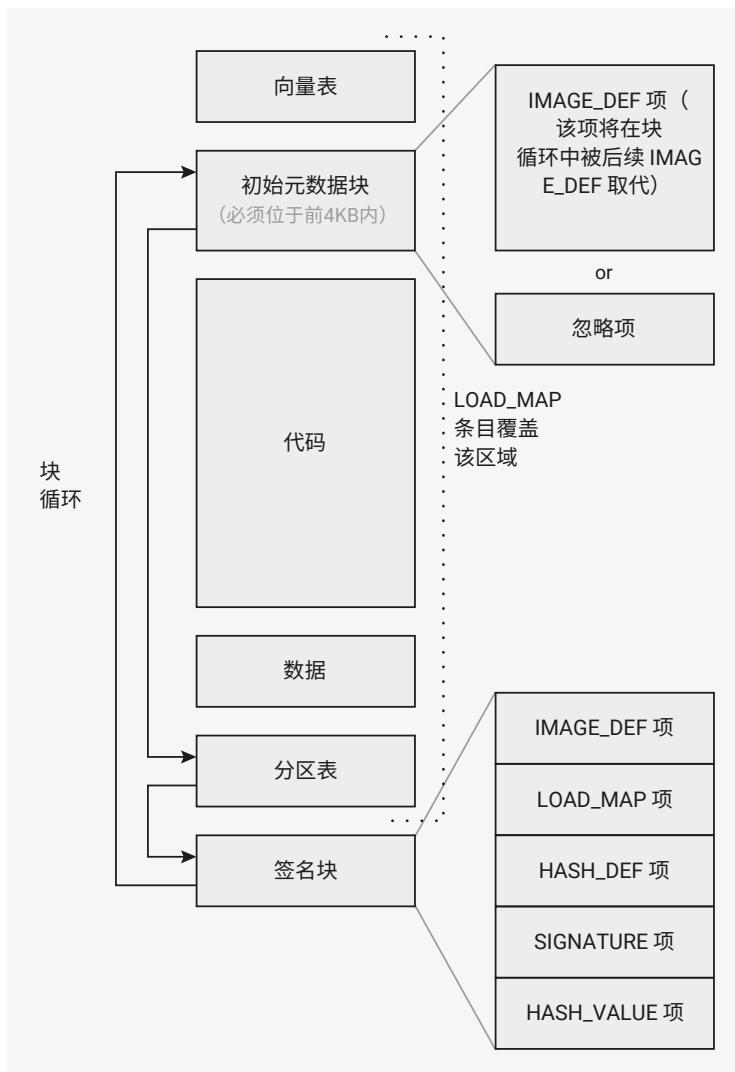
一个示例闪存布局可能如下所示：

```

槽0 (0x00000000-0x00008000)
引导加载程序映像0
分区表0
槽1 (0x00008000-0x00010000)
引导加载程序映像1
分区表1
分区0 (0x00010000-0x00020000)
二进制A
分区1 (0x00020000-0x00030000)
链接类型："A"
链接值：0
二进制 B

```

包含 `IMAGE_DEF` 和 `PARTITION_TABLE` 的块循环在此情况下（签名后）可能如下所示：



请注意块循环中的三个块：

1. 前 4 kB 内的原始块（内容无关紧要，因稍后的 **IMAGE\_DEF** 将覆盖其内容）
2. 二进制文件末尾的 **PARTITION\_TABLE**
3. 已签名的 **IMAGE\_DEF**

#### **注意**

可以分别对 **PARTITION\_TABLE** 和 **IMAGE\_DEF** 进行签名，然而为实现最快启动速度，bootrom 允许在 **IMAGE\_DEF** 中使用“覆盖型” **LOAD\_MAP**。只要 **LOAD\_MAP** 中定义的进行哈希/签名的区域涵盖整个 **PARTITION\_TABLE** 块，“覆盖”签名即可用于验证该 **PARTITION\_TABLE**。

为了使引导加载程序找到并启动新的映像，可能需要利用各种 bootrom 方法：

- **get\_partition\_table\_info()** 用于获取完整的分区信息。
- 或者调用 **get\_partition\_table\_info()**，传入 **SINGLE\_PARTITION**、所选分区号及 **PARTITION\_LOCATION\_AND\_FLAGS**，以获取单个分区的地址。

```

uint32_t partition_info[3];
get_partition_table_info(partition_info, 3, PT_INFO_PARTITION_LOCATION_AND_FLAGS
 | PT_INFO_SINGLE_PARTITION | (boot_partition << 24));
uint16_t first_sector_number = (partition_info[1]
 & PICOBIN_PARTITION_LOCATION_FIRST_SECTOR_BITS)
 >> PICOBIN_PARTITION_LOCATION_FIRST_SECTOR_LSB;
uint16_t last_sector_number = (partition_info[1]
 & PICOBIN_PARTITION_LOCATION_LAST_SECTOR_BITS)
 >> PICOBIN_PARTITION_LOCATION_LAST_SECTOR_LSB;
uint32_t data_start_addr = first_sector_number * 0x1000;
uint32_t data_end_addr = (last_sector_number + 1) * 0x1000;
uint32_t data_size = data_end_addr - data_start_addr;

```

- 使用 `get_sys_info()` 携带 `BOOT_INFO` 参数以获取 `flash_update_boot_window_base` (如有) :

```

uint32_t sys_info[5];
get_sys_info(sys_info, 5*4, SYS_INFO_BOOT_INFO);
uint32_t flash_update_boot_window_base = sys_info[3];

```

- 使用 `pick_ab_partition()` 在 A/B 分区间选择启动分区 (如需要) :

```

uint8_t boot_partition = pick_ab_partition(workarea, 0xC00, 0,
 flash_update_boot_window_base);

```

- 或者使用 `get_b_partition()` 直接定位另一个分区。
- 使用 `chain_image()` 携带 `data_start_addr` 和 `data_size` 参数以启动所选映像:

```

// 注意, 第三个参数为负数表示告知 chain_image 该映像作为“闪存更新”启动流程的一部分被链式调用,
// 且不能直接从 SRAM 启动

(XIP_BASE + data_start_addr) * (info.boot_type == BOOT_TYPE_FLASH_UPDATE ? -1 :
1),
data_size
);

```

### 注意

所使用的 `workarea` 不得与被链入的映像重叠, 因此须注意 SRAM 或封装的二进制文件。若二进制文件与 `workarea` 重叠, 结果不可预测, 但极有可能是不良后果。

## 5.10.7. OTP 引导加载程序

此情形类似于自定义引导加载程序, 但程序将存储于 OTP 并在 SRAM 中运行。

一种可能的应用是将解密代码置于 OTP, 从而将闪存分区的可执行映像解密至 RAM 中。

整个引导加载程序必须位于 OTP 行范围 `0x0C0` 至 `0xF48` 内, 以避免干扰其他保留 OTP 功能, 最大尺寸为 7440 字节 (每个 ECC 行 2 字节)。若部分引导密钥及 OTP 密钥未被使用, 该区域可在任一端适度延伸。

OTP引导加载程序须以ECC格式存储，起始于OTPBOOT\_SRC指定的行，大小由OTPBOOT\_LEN确定。启动时，将加载至OTPBOOT\_DST0和OTPBOOT\_DST1指定的地址，且该地址必须位于主SRAM内。引导加载程序必须符合标准映像的条件：必须包含IMAGE\_DEF，且启用安全启动时必须签名。

一旦OTP引导加载程序写入OTP，并且设置了OTPBOOT\_SRC、OTPBOOT\_LEN、OTPBOOT\_DST0和OTPBOOT\_DST1，即可通过设置BOOT\_FLAGS0.ENABLE OTP\_BOOT启用OTP引导。如果OTP映像未通过bootrom的启动检查，默认情况下，引导将继续走正常的闪存引导路径。您可以通过设置BOOT\_FLAGS0.DISABLE\_FLASH\_BOOT来阻止此情况发生。

#### ● 警告

编写OTP引导加载程序时务必极为谨慎。ECC行一经写入，将无法修改。

### 5.10.8. 回滚版本及引导加载程序

#### ● 警告

忽视本节建议可能导致您的设备无法启动。

对于需链入带回滚版本的可执行映像的安全RP2350引导加载程序，您必须使用单独的OTP行分别用于：

- 引导加载程序的回滚版本
- 链入的可执行映像的回滚版本

否则，提升链入可执行映像版本将导致OTP引导加载程序及设备无法启动。

您还必须确保引导加载程序和可执行映像的回滚版本均为非零，因为相关要求回滚版本的OTP标志是全局性的。未遵守将导致设备无法启动。

我们建议使用DEFAULT\_BOOT\_VERSION0和DEFAULT\_BOOT\_VERSION1行作为二进制文件的回滚版本，并在OTP中选择其他未使用的行作为引导程序的回滚版本。

# 第6章 电源

## 6.1. 电源供应

RP2350 需要五个独立的电源供应。但在大多数应用中，可以将其中几个合并接入单一电源。典型应用通常只需要单一的 3.3 V 电源。见图19。

以下章节描述了电源供应及若干潜在的电源供应方案。详细的电源供应参数见第14.9.5节。

### 6.1.1. 数字 IO 电源 ( IOVDD )

**IOVDD** 为芯片GPIO提供IO电源，标称电压应为1.8 V至3.3 V之间。

供电电压决定数字IO的外部信号电平，应根据所需电平选择，详见第14.9节。所有GPIO共用同一电源，并以相同信号电平工作。

若数字IO以标称1.8 V供电，输入阈值应通过将VOLTAGE\_SELECT寄存器设置为 1 进行调整。VOLTAGE\_SELECT默认设置为 0，此时输入阈值适用于标称IO电压介于2.5 V至3.3 V之间。详见第9章。

#### ⚠ 警告

IOVDD以1.8 V供电且输入阈值设定为2.5 V至3.3 V的供电范围，此为安全操作模式，但输入阈值不符合规范要求。若IO以高于1.8 V的电压供电，且输入阈值设定为1.8 V供电范围，可能导致芯片损坏。

### 6.1.2. QSPI IO 电源 ( QSPI\_IOVDD )

**QSPI\_IOVDD**为芯片的**QSPI**接口提供**IO**电源，供电电压应为标称1.8 V至3.3 V之间。该供电电压决定**QSPI**接口的外部信号电平，应根据所需电平选取，详见第14.9节。在大多数应用中，**QSPI**接口将连接至外部闪存设备，该设备决定所需的信号电平。

若**QSPI**接口标称供电为1.8 V，应通过将VOLTAGE\_SELECT寄存器设置为 1 以调整IO输入阈值。VOLTAGE\_SELECT默认设置为 0，此时输入阈值适用于标称IO电压介于2.5 V至3.3 V之间。详见第9章。

#### ⚠ 警告

IOVDD以1.8 V供电且输入阈值设定为2.5 V至3.3 V的供电范围，此为安全操作模式，但输入阈值不符合规范要求。若IO以高于1.8 V的电压供电，且输入阈值设定为1.8 V供电范围，可能导致芯片损坏。

### 6.1.3. 数字核心电源 ( DVDD )

芯片核心数字逻辑由 **DVDD** 供电，标称电压为1.1 V。片内专用核心电压调节器允许将2.7 V至5.5 V的输入电源转换为 **DVDD**。详见第6.3节。另可由片外电源直接供给 **DVDD**。

若使用片内核心电压调节器，靠近调节器的两引脚 **DVDD** 应使用100nF电容器靠近引脚去耦。距离调节器最远的 **DVDD** 引脚应使用4.7μF电容器靠近引脚去耦。

引脚。

#### 6.1.4. USB PHY 与 OTP 电源 (USB\_OTP\_VDD)

USB\_OTP\_VDD为芯片的USB PHY及OTP只读存储器供电，应以标称3.3 V供电。为减少外部电源数量，USB\_OTP\_VDD可与核心电压调节器的模拟电源(VREG\_AVDD)或数字IO电源(IOVDD)共用同一电源，前提是IOVDD同样为3.3 V供电。该电源必须始终提供，即便在USB PHY未使用的应用中亦然。

USB\_OTP\_VDD应在芯片的USB\_OTP\_VDD引脚附近通过100nF电容进行去耦。

#### 6.1.5. ADC 电源 (ADC\_AVDD)

ADC\_AVDD为芯片的模数转换器(ADC)供电。其标称供电电压范围为1.8 V至3.3 V，但电压低于2.97 V时，ADC性能将受影响。为减少外部电源数量，ADC\_AVDD可与核心电压调节器的模拟电源共用同一电源。

(VREG\_AVDD) 或数字IO电源(IOVDD)。

##### ① 注意

向ADC\_AVDD供电时，电压高于或低于IOVDD是安全的，例如，为实现最佳性能，可以以3.3 V为ADC供电，同时支持数字IO的1.8 V信号电平。但ADC模拟输入端的电压不得超过IOVDD，例如若IOVDD为1.8 V，ADC输入端电压应限制在1.8 V以内。超过IOVDD的电压将导致通过ESD保护二极管的漏电流发生。详情请参见第14.9节。

应在芯片的ADC\_AVDD引脚附近使用一个100nF电容对ADC\_AVDD进行去耦。

#### 6.1.6. 核心电压稳压器输入电源 (VREG\_VIN)

VREG\_VIN为片上核心电压调节器的输入电源，电压范围应为2.7 V至5.5 V。为减少外部电源数量，VREG\_VIN可与电压调节器模拟电源(VREG\_AVDD)或数字IO电源(IOVDD)共用同一电源。应注意尽量减少VREG\_AVDD上的噪声。

应在VREG\_VIN与地之间连接一个4.7 μF电容，且该电容应靠近芯片的VREG\_VIN引脚。

有关片上电压调节器的更多详情，请参见第6.3节。

#### 6.1.7. 芯片内电压稳压器模拟电源 (VREG\_AVDD)

VREG\_AVDD为片上电压调节器的模拟控制电路供电，额定供电电压应为3.3 V。

为减少外部电源数量，VREG\_AVDD可使用与电压调节器输入电源(VREG\_VIN)或数字IO电源(IOVDD)相同的电源。应注意尽量减少VREG\_AVDD上的噪声。

可能需要被动低通滤波器，详情请参见第6.3.7节。

##### ① 注意

VREG\_AVDD还为芯片的上电复位和欠压检测模块供电，因此即使不使用片上电压调节器，也必须为其供电。

### 6.1.8. 电源供应顺序

除两个电压调节器供电输入（**VREG\_VIN**和**VREG\_AVDD**）应同时上电外，RP2350的电源可按任意顺序上电或断电。但是，如果ADC电源（**ADC\_AVDD**）在数字核心电源（DVDD）之前上电，或在数字核心电源之后断电，可能会有小的瞬态电流流动。这不会损坏芯片，但可通过先于或同时为**ADC\_AVDD**供电后再为DVDD供电，以及在断开**ADC\_AVDD**电源时，同时或随后断开DVDD电源来避免。在最常见的电源方案中，当芯片由单一3.3 V电源供电时，由于片上电压稳压器的启动时间，DVDD的上电时间会稍晚于**ADC\_AVDD**。此行为属可接受情况。

## 6.2. 电源管理

RP2350保留了RP2040的电源控制功能，并通过将芯片的数字核心划分为多个可选择性断电的电源域加以扩展，从而在芯片非持续激活的应用场景中实现显著的功耗节约。本节描述核心电源域及其控制方式。传统的RP2040电源控制功能仍提供有效的节能效果，详见第6.5节。

电源域及电源状态之间的切换由电源管理器控制。电源管理器运行于内部低功耗振荡器 **lposc** 或参考时钟 **clk\_ref**。设备可通过软件控制配置以关闭电源，并可通过GPIO或定时器事件唤醒。电源管理器的配置通过第6.4节中的**POWMAN**寄存器完成。

### 6.2.1. 核心电源域

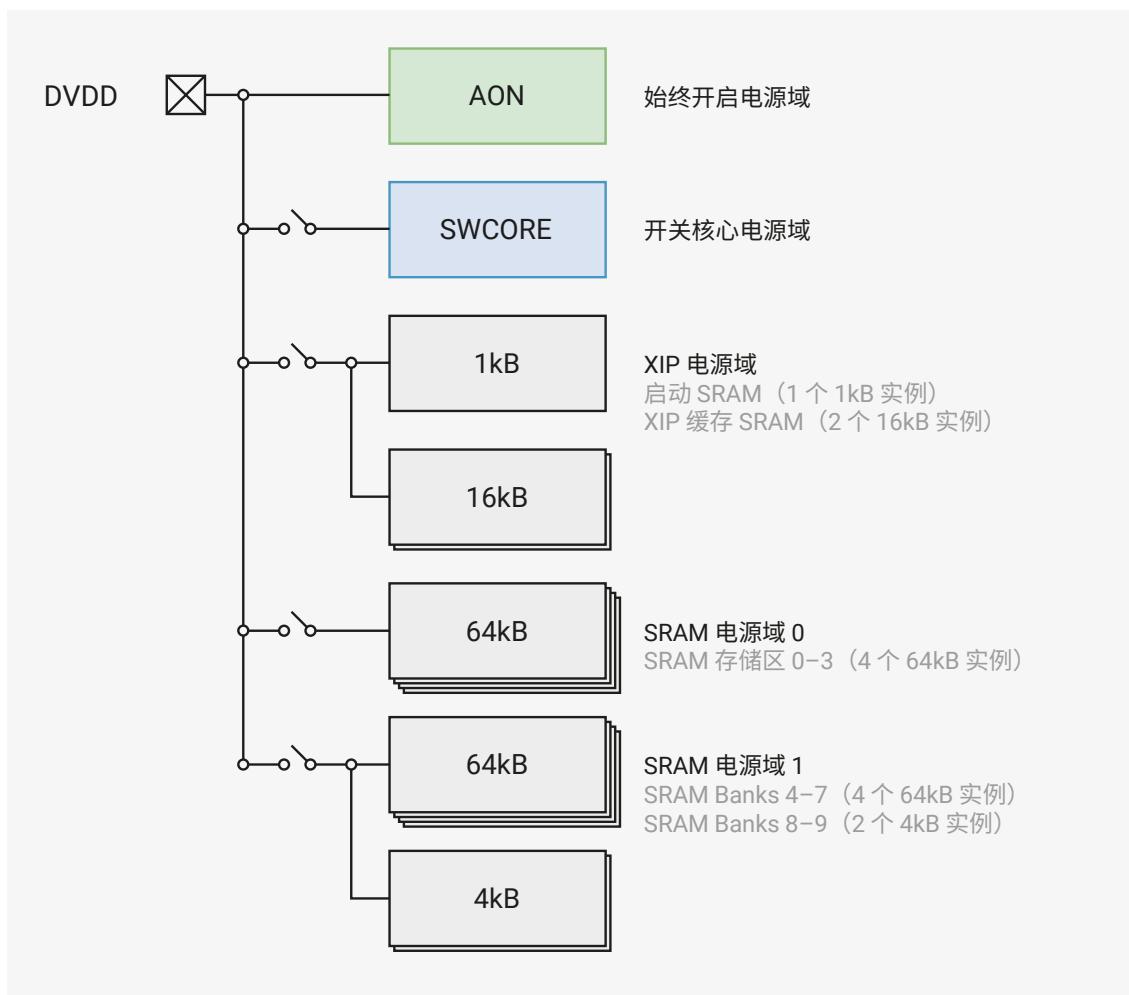
RP2350的核心逻辑划分为五个电源域。在某些限制条件下，这些电源域可以选择性关闭，以降低芯片功耗。五个电源域为：

- **AON** - 始终开启 - 芯片核心供电（**DVDD**）可用时始终保持供电的小部分逻辑
- **SWCORE** - 切换核心 - 其余核心逻辑功能，包括处理器、总线结构、外设等。
- **XIP** - XIP 缓存 SRAM 及启动 RAM
- **SRAM0** - SRAM 电源域 0 — 大型 SRAM 存储区的下半部分
- **SRAM1** - SRAM 电源域 1 — 大型 SRAM 存储区的上半部分及临时 SRAM

始终开启（AON）域中的逻辑控制其他电源域的供电状态，这些电源域可独立开关电源。唯一例外为 XIP 域，当 SWCORE 域通电时，XIP 域必须始终保持供电。被供电的 SRAM 在开关核心断电时内容将被保留。

图 18 概述了核心电源域架构。

图 18. 核心电源域



### 6.2.2. 电源状态

RP2350 可根据各电源域的开启或关闭状态处于多个电源状态。电源状态的命名格式为 **Pc.m**，其中：

- **c** 表示切换核心 (SWCORE) 电源域的状态：**0**= 开启 / **1**= 关闭
- **m** 为内存电源域的 3 位二进制表示，顺序为 XIP、SRAM0、SRAM1

**P0.m** 状态表示切换核心处于开启状态，为正常操作状态。**P1.m** 状态表示切换核心处于关闭状态，为低功耗状态。

表 475 显示了所有可用电源状态。

表475. 支持的电源状态

电源状态	描述	AON	SWCORE	XIP	SRAM0	SRAM1
P0.0	正常操作	开启	开启	开启	开启	开启
P0.1	正常操作 (SRAM1 关闭)	开启	开启	开启	开启	关闭
P0.2	正常操作 (SRAM0 关闭)	开启	开启	开启	关闭	开启
P0.3	正常操作 (SRAM0 和 SRAM1 关闭)	开启	开启	开启	关闭	关闭
P1.0	低功耗	开启	关闭	开启	开启	开启
P1.1	低功耗 (SRAM1 关闭)	开启	关闭	开启	开启	关闭
P1.2	低功耗 (SRAM0 关闭)	开启	关闭	开启	关闭	开启

电源状态	描述	AON	SWCORE	XIP	SRAM0	SRAM1
P1.3	低功耗 (SRAM0 和 SRAM1 关闭)	开启	关闭	开启	关闭	关闭
P1.4	低功耗 (XIP 关闭)	开启	关闭	关闭	开启	开启
P1.5	低功耗 (XIP 和 SRAM1 关闭)	开启	关闭	关闭	开启	关闭
P1.6	低功耗 (XIP 和 SRAM0 关闭)	开启	关闭	关闭	关闭	开启
P1.7	低功耗 (XIP、SRAM0 和 SRAM1 关闭)	开启	关闭	关闭	关闭	关闭
关闭	无电源	关闭	关闭	关闭	关闭	关闭

在关闭状态下，芯片无外部电源且所有域均无电。芯片在接通外部电源后，会自动从关闭状态切换至 P0.0 状态。

要确定当前电源状态，请读取 STATE.CURRENT 字段。 CURRENT 是一个 4 位字段，表示切换核心和内存电源域的电源状态。

### 6.2.3. 电源状态转换

功率状态之间的转换可由软件、硬件或通过芯片的调试子系统发起。启动后，转换由芯片 AON 功率域内的自主功率序列器管理。功率序列器可通过 SEQ\_CFG 寄存器进行有限配置。同样，可通过芯片调试子系统中的 RP\_AP 寄存器对序列器进行有限观察和控制。这些寄存器详述于第 3.5.10 节。

有效的功率状态转换如下：

- 所有从一个 P0.m 状态（已开启切换核心）到另一个 P0.m 状态（已开启切换核心）的转换，若该转换增加或减少了开启的 SRAM 域数量
- 所有从 P0.m 状态（已开启切换核心）到 P1.m 状态（已关闭切换核心）的转换，除非该转换导致已关闭的 SRAM 域被开启
- 所有从 P1.m 状态（切换核心电源关闭）向 P0.m 状态（切换核心电源开启）的转换，除导致开启的 SRAM 域变为关闭的转换

从一个 P1.m 状态（切换核心电源关闭）转换至另一个 P1.m 状态（切换核心电源关闭）的操作不被支持，且硬件将禁止此类转换。

有效转换如下表所示。

表476。有效电源状态转换

起始状态	目标状态													
P0.0		P0.1	P0.2	P0.3	P1.0	P1.1	P1.2	P1.3	P1.4	P1.5	P1.6	P1.7		
P0.1	P0.0			P0.3		P1.1		P1.3		P1.5		P1.7		
P0.2	P0.0			P0.3			P1.2	P1.3			P1.6	P1.7		
P0.3	P0.0	P0.1	P0.2					P1.3				P1.7		
P1.0	P0.0													
P1.1	P0.0	P0.1												
P1.2	P0.0		P0.2											
P1.3	P0.0	P0.1	P0.2	P0.3										
P1.4	P0.0													
P1.5	P0.0	P0.1												

起始状态	目标状态										
P1.6	P0.0		P0.2								
P1.7	P0.0	P0.1	P0.2	P0.3							

### 6.2.3.1. 从正常运行 (P0.m) 状态的转换

从正常运行状态 ( $P0.m$ ) 向低功耗状态 ( $P1.m$ ) 或另一个正常运行状态 ( $P0.m$ ) 的转换，需通过写入STATE.REQ字段来触发。REQ为4位字段，表示切换核心及存储电源域所请求的电源状态。STATE.WAITING字段将被立即设置，随后在实际状态变更开始时设置STATE.CHANGING字段。如果请求切换到低功耗 ( $P1.m$ ) 状态，WAITING字段将保持设置，直到处理器通过 `_wfi()` 进入低功耗状态。在 WAITING状态下，写入 STATE.REQ 字段可更改或取消初始请求。在 CHANGING 状态时，无法更改请求的状态。

请求切换到不支持的状态或导致无效转换的状态时，STATE.BAD\_SW\_REQ 字段将被设置。

如果在 WAITING状态下接收到硬件上电请求，则通过 STATE.REQ 请求的转换将被中止，硬件上电请求将被完成。STATE.PWRUP WHILE\_WAITING 和 STATE.REQ IGNORED 字段将被设置。

写入 STATE.REQ 时：

- 若存在待处理的上电请求，则设置STATE.REQ IGNORED，且不采取后续操作。
- 若请求的状态无效，则设置STATE.BAD\_SW\_REQ，且不采取后续操作。
- 若被切换的核正处于断电状态，则设置STATE.WAITING，直至两个处理器均进入 `_wfi()` 状态。随后将设置STATE.CHANGING，然而此时不会对任何处理器进行上电以读取该标志。
  - 若在STATE.WAITING期间出现上电请求，则设置STATE.PWRUP WHILE\_WAITING，该状态亦可触发中断以使处理器退出 `_wfi()` 状态。不采取任何后续操作。
  - 在两个处理器进入 `_wfi()` 之前，通过向STATE.REQ写入新请求，可使系统退出WAITING状态。
- 任何非断电切换核的状态请求，如对SRAM域0或1进行上电或断电，将立即启动。软件应等待直到 STATE.CHANGING 标志清除，以了解断电顺序。  
在 STATE.CHANGING 标志清除后，STATE.CURRENT 会被更新。
- 如果是上电，软件也应等待 STATE.CHANGING 标志，以确保所有部分均已上电，然后再继续。在实际操作中，该过程由 RP2350 bootrom 处理。

无效的状态转换包括：

- 任何上电与断电请求的组合
- 任何导致 XIP/bootRAM 断电且 SWCORE 上电的请求

若 XIP、boot RAM、sram0 或 sram1 在 SWCORE 断电时仍保持上电，sram 会自动切换至低功耗状态。存储的数据将被保留。

在切换至切换核心断电状态 ( $P1.m$ ) 之前，软件需配置：

- 如有需要，配置 GPIO 唤醒条件
- 如有需要，配置唤醒闹钟
- SRAM0 和 SRAM1 区域的返回状态

### 6.2.3.2. 从低功耗 (P1.m) 状态的转换

从 P1.m 状态切换到 P0.m 状态由 GPIO 事件或定时器警报触发。

最多有 5 个唤醒源：

- 最多 4 个 GPIO 唤醒（高电平/低电平或下降沿/上升沿）
- 1 个警报唤醒

GPIO 唤醒通过 PWRUP0-PWRUP3 寄存器进行配置。在电源序列器完成关机操作之前，唤醒功能不会启用。

警报唤醒通过写入 ALARM\_TIME\_15T00 至 ALARM\_TIME\_63T048 寄存器进行配置。警报唤醒的时间分辨率为 1 毫秒。设置完成后，警报唤醒通过同时向 TIMER.PWRUP\_ON\_ALARM 和 TIMER.ALARM\_ENAB 写入 1 来启动。如果在关机序列期间警报触发，电源上电序列将在关机序列完成后启动。

LAST\_SWCORE\_PWRUP 寄存器指示最近一次上电的触发事件。

### 6.2.3.3. 调试器启动的电源状态转换

调试器可通过 SW-DP CTRL/STAT 寄存器的 CSYSPWRUPREQ 输出触发上电序列。

此操作将为所有域供电（即返回至状态 P0.0），并禁止任何进一步的软件发起的电源状态转换。

当 CSYSPWRUPREQ 被置位时，电源序列控制器将执行以下操作：

- 完成任何正在进行的电源状态转换。
- 返回至电源状态 P0.0。
- 置位 CSYSPWRUPACK 以向调试主机发送完成信号。

如果 CSYSPWRUPREQ 被取消置位，则允许软件发起的电源转换恢复。用户可以通过以下提示检测因 CSYSPWRUPREQ 导致的软件请求转换被忽略的情况：

- 写入 STATE.REQ 后出现 STATE.REQ\_IGNORED。
- CURRENT\_PWRUP\_REQ 的第 5 位（coresight）被置位。
- 以下任一情况：
  - 使调试器取消置位 CSYSPWRUPREQ，或
  - 通过设置 DBG\_PWRCFG.IGNORE 屏蔽 CSYSPWRUPREQ

#### **注意**

DBG\_PWRCFG.IGNORE 对于测试连接调试器时进入休眠或忽略 CSYSPWRUPREQ 非常有用。调试器断开时，通常会保持 CSYSPWRUPREQ 为设定状态。若无 DBG\_PWRCFG.IGNORE，之后无法进入休眠。

### 6.2.3.4 电源模式感知 GPIO 控制

电源管理序列器能够在进入及退出 P1.m 状态时切换两个 GPIO 输出的状态，即切换核心处于断电状态时。该功能使外部设备能够感知电源状态。核心断电后，GPIO 切换以指示低功耗状态；核心通电前，GPIO 切换以指示高功耗状态，确保外部组件的高功耗状态始终与核心的高功耗状态重叠。GPIO 由 EXT\_CTRL0 和 EXT\_CTRL1 寄存器配置。

### 6.2.3.5. 隔离

关闭 SWCORE 电源时，垫片控制信号和数据信号将被锁存并与 IO 逻辑隔离。此举可避免垫片产生可能损坏外部元件的信号切换。SWCORE 加电时，隔离不会自动解除。用户须在配置 IO 逻辑后，通过清除垫片控制寄存器（例如 `GPIO0.ISO`）中的 ISO 字段以解除隔离。

## 6.3. 核心电压调节器

RP2350 为其实数核心供电 (`DVDD`) 提供片上电压调节器。该调节器要求输入电压为 2.7 V 至 5.5 V (`VREG_VIN`)，可直接由单节锂离子电池或 USB 电源产生 `DVDD`。调节器的模拟控制电路需要单独的标称 3.3 V 低噪声电源 (`VREG_AVDD`)。该调节器支持开关和线性两种调节模式，确保在高负载和低负载状态下均能高效工作。

为使芯片启动，调节器默认启用，并将在其电源可用时立即供电。

调节器以开关模式启动，标称输出电压为 1.1 V，但芯片复位解除后，其工作模式和输出电压可被更改。输出电压可设置在 0.55 V 至 3.30 V 范围内，调节器最大供应电流为 200mA。

虽然调节器设计用于芯片数字核心电源 (`DVDD`)，但若 `DVDD` 由外部电源直接供电，调节器亦可用于其他用途。

### 6.3.1. 运行模式

调节器具备以下三种工作模式。

#### 6.3.1.1. 正常模式

在正常模式下，调节器以开关模式运行，最大供应电流为 200mA。正常模式用于 `P0.x` 电源状态，即芯片切换核心电源开启时。调节器必须在允许核心电源电流超过 1mA 之前处于正常模式。当调节器的输入电源首次接通时，调节器以正常模式启动。

#### 6.3.1.2. 低功耗模式

在低功耗模式下，调节器以线性模式工作，最大输出电流仅为 1mA。低功耗模式适用于 `P1.x` 电源状态，此时芯片的切换核心处于断电状态。调节器进入低功耗模式前，核心供电电流必须低于 1mA。调节器在低功耗模式下的输出电压限制为 1.3 V。

#### ⚠ 警告

在低功耗模式下，调节器的输出直接连接至 `DVDD`。在此模式下，无法将调节器从 `DVDD` 断开。如果 `DVDD` 由外部电源供电，则不得将调节器置于低功耗模式。

#### 6.3.1.3. 高阻抗模式

在高阻抗模式下，调节器被禁用，功耗降至最低，输出端置于高阻抗状态。仅当数字核心电源 (`DVDD`) 由外部调节器供电时，才应使用此模式。

调节器。如果片内调节器向DVDD供电，进入高阻抗模式将触发复位事件，使片内调节器恢复到正常模式。

### 6.3.2. 软件控制

#### ● 警告

调节器解锁后不可重新上锁。避免对VREG寄存器的误写操作。

调节器可由软件直接控制，但须先向VREG\_CTRL寄存器的UNLOCK字段写入1以完成解锁。解锁后，可通过VREG寄存器对调节器进行控制。

调节器的工作模式在初次上电或复位后默认为正常模式，但可通过向VREG寄存器的HIZ字段写入1切换至高阻抗模式。调节器的输出电压可通过写入寄存器的VSEL字段进行设置，具体可用设置详见VREG寄存器说明。为防止意外过压，输出电压被限制在1.3 V，除非VREG\_CTRL中的DISABLE\_VOLTAGE\_LIMIT字段被设置。输出电压在初次上电或复位事件后默认为1.1 V。

当调节器的工作模式或输出电压正在更新时，VREG寄存器中的UPDATE\_IN\_PROGRESS字段将被设置。当UPDATE\_IN\_PROGRESS字段被设置时，对寄存器的写入操作将被忽略。

由于软件运行时负载电流将超过1mA，因此无法通过软件控制将调节器置于低功耗模式。

#### ⚠ 警告

调节器的输出电压可在0.55 V至3.3 V之间调节，但RP2350的数字核心电源（DVDD）在非1.1 V电压下可能无法可靠运行。

### 6.3.3. 电源管理器控制

调节器的工作模式和输出电压也可以由电源管理器控制。电源管理器控制通常应用于芯片进入或退出低功耗（P1.x）状态且软件可能未运行的情况。

除正常模式和高阻抗模式外，电源管理器控制还允许调节器进入低功耗模式。默认情况下，调节器在进入低功耗（P1.x）状态时切换至低功耗模式，返回正常（P0.x）状态时切换回正常模式。

低功耗状态下的工作模式和输出电压由VREG\_LP\_ENTRY寄存器中的值设定。

芯片返回正常状态时所采用的工作模式和输出电压由VREG\_LP\_EXIT寄存器中的值设定。寄存器中包含一个额外的MODE字段，用以选择低功耗模式。

必须在请求转换至低功耗状态之前由软件写入寄存器值，因为转换期间及之后软件将不运行。实际的低功耗状态转换由电源管理器负责处理。芯片恢复到正常状态后，软件即可运行并直接控制调节器。VREG寄存器的数值反映了芯片恢复正常状态后，稳压器当前的工作模式和输出电压。

### ⚠ 警告

低功耗模式应仅在稳压器为芯片数字核心电源 (**DVDD**) 供电时使用，因为稳压器的低功耗输出连接至芯片上的 **DVDD** 端。

### 6.3.4. 状态

要确定稳压器的状态，请读取包含两个字段的VREG\_STS寄存器：

- **VOUT\_OK**指示电压稳压器的输出电压是否被正确调节。上电时，**VOUT\_OK**保持低电平，直到稳压器启动且输出电压达到**VOUT\_OK**断言阈值 (**VOUT\_OK<sub>TH.ASSERT</sub>**)。随后，**VOUT\_OK**保持高电平，直到电压降至**VOUT\_OK**取消断言阈值 (**VOUT\_OK TH.DEASSERT**) 以下，此后保持低电平，直到输出电压再次超过断言阈值。**VOUT\_OK TH.ASSERT**名义值为所选输出电压的90%；若所选输出电压为1.1 V，则该值为0.99 V。**VOUT\_OKTH.DEASSERT**名义值为所选输出电压的87%；若所选输出电压为1.1 V，则为0.957 V。详见第14.9.6节。
- 当稳压器启动时，**STARTUP**信号为高电平，并保持高电平，直到通过软件或电源管理器更改稳压器的工作模式或输出电压。

将输出电压调整至更高电压时，**VOUT\_OK**信号将变为低电平，直至达到该更高电压的断言阈值。如果稳压器处于高阻抗模式，**VOUT\_OK**信号同样会变为低电平。

### 6.3.5. 电流限制

电压稳压器包含电流限制功能，以防止负载电流超过最大额定值。当电流限制启动时，输出电压将失去调节并低于所选值。详见第14.9.6节。

### 6.3.6. 过温保护

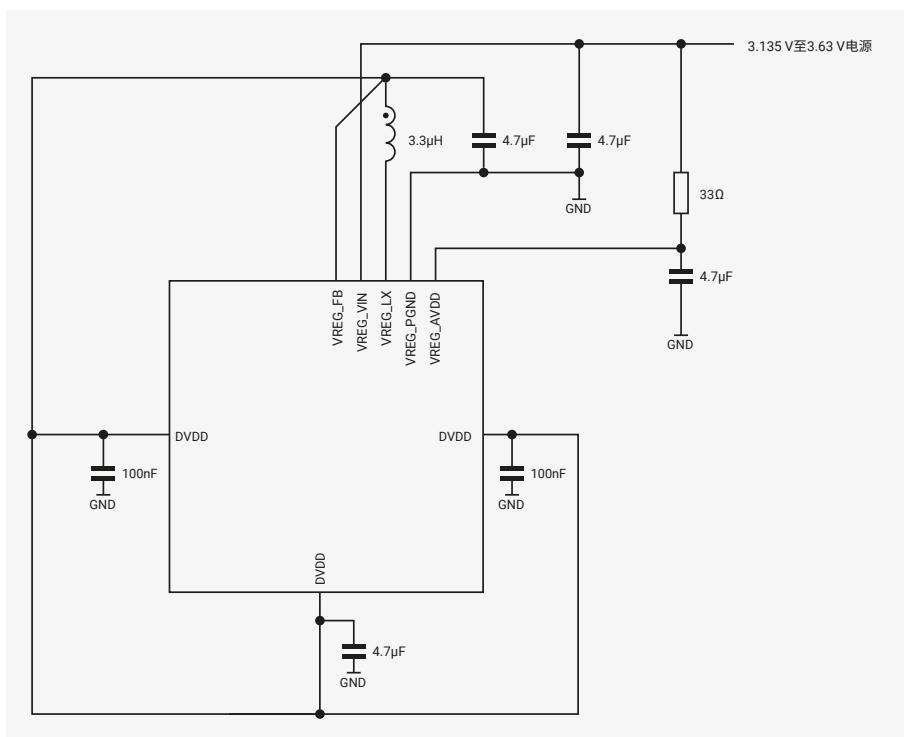
若晶体管结温超过VREG\_CTRL寄存器中 **HT\_TH** 字段设定的阈值，稳压器将终止调节并关闭其功率晶体管。当晶体管结温下降至该温度阈值以下约20°C时，稳压器将恢复调节。

### 6.3.7. 应用电路

稳压器需要两个外部电源，输入电源 (**VREG\_VIN**) 和供模拟控制电路的独立低噪声电源 (**VREG\_AVDD**)。**VREG\_VIN**必须在2.7 V至5.5 V范围内，且**VREG\_AVDD**必须在3.135 V至3.63 V范围内。

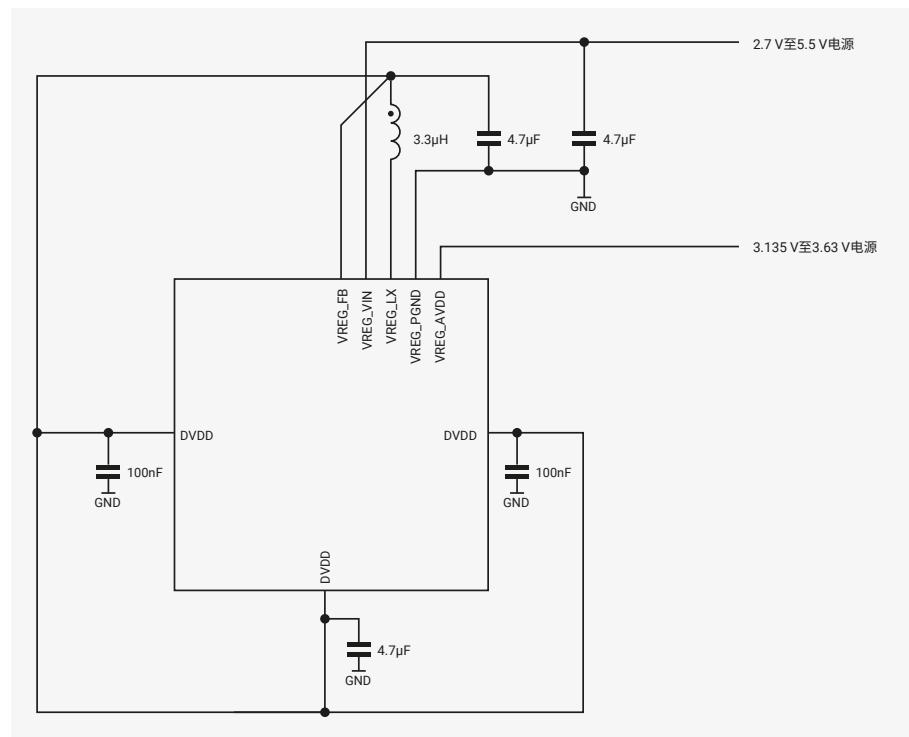
如果 **VREG\_VIN** 被限制在3.135 V至3.63 V范围内，则可以使用单一组合电源为**VREG\_VIN**和**VREG\_AVDD**供电。该方案如图19所示。请注意最大限度地减少 **VREG\_AVDD** 上的噪声。

图19 带组合  
电源的核心电压调  
节器



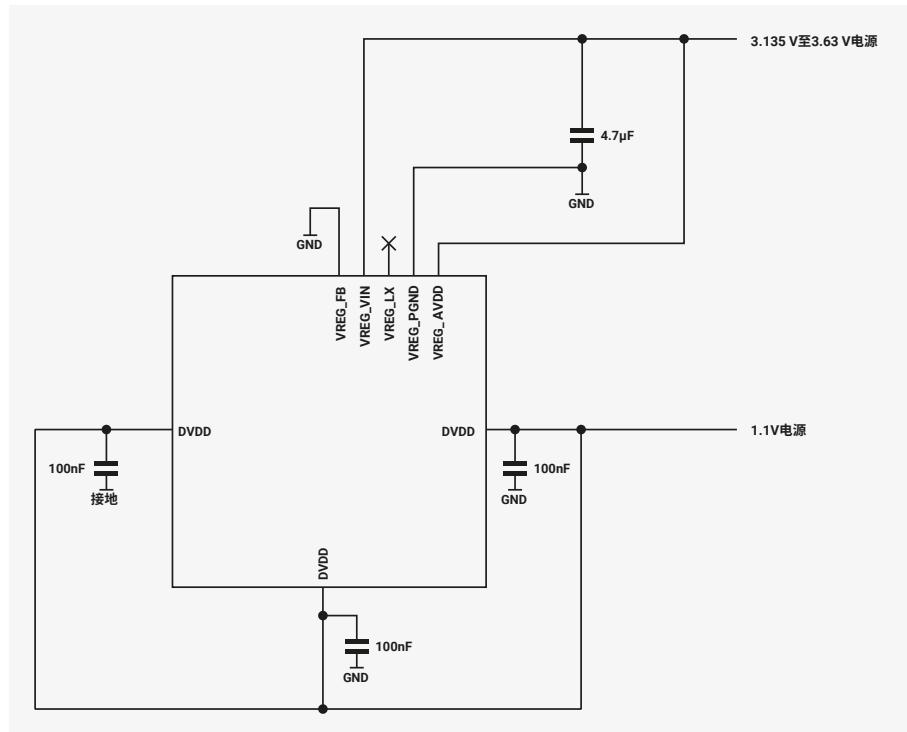
或者，为支持高于3.63 V的输入电压，**VREG\_VIN**和**VREG\_AVDD**可分别供电。如图20所示。

图20。带独  
立电源的核心电压  
调节器



若数字核心供电（DVDD）由外部1.1V电源提供，片上调节器可被禁用，从而简化应用电路。但仍需为调节器的模拟电源（**VREG\_AVDD**）及输入电源（**VREG\_VIN**）供电，以驱动芯片的上电复位和欠压检测电路。电感器可省略，仅需一个输入电容。将**VREG\_FB**直接接地。如图21所示。

图21。禁用片上  
调节器的外部核  
心供电。



只要 `VREG_VIN` 和 `VREG_AVDD` 可用，片上调节器将自动上电；但芯片复位完成后，可通过软件控制关闭其工作。这是安全的操作模式，但稳压器在关闭之前将消耗大约  $400 \mu\text{A}$  的电流。应通过向 `VREG` 寄存器的 `HIZ` 字段写入 1 来关闭稳压器。

### 6.3.8. 外部元件及PCB布局要求

RP2350 PCB 布局中最关键的部分是核心电压稳压器。该部分应优先放置于任何电路板设计中，且必须严格遵守相关指导原则。

图 22。Raspberry  
Pi Pico 2 原  
理图中的稳压器  
部分。加粗显示  
的网络表示高开  
关电流路径  
。

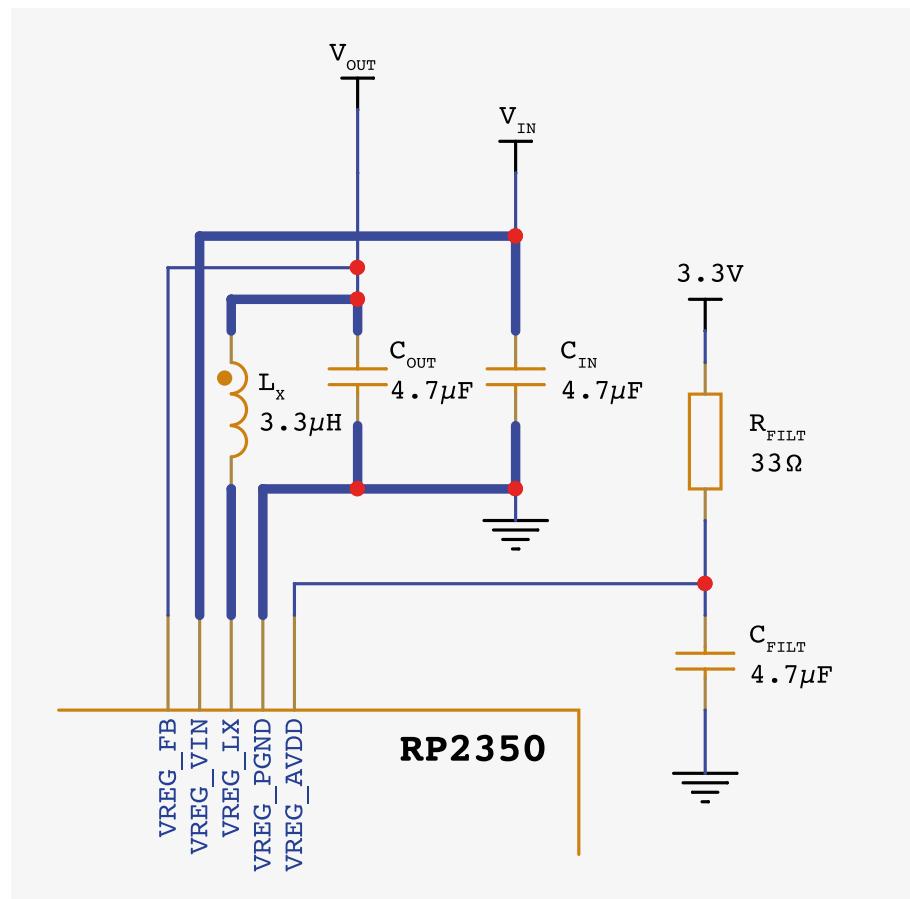
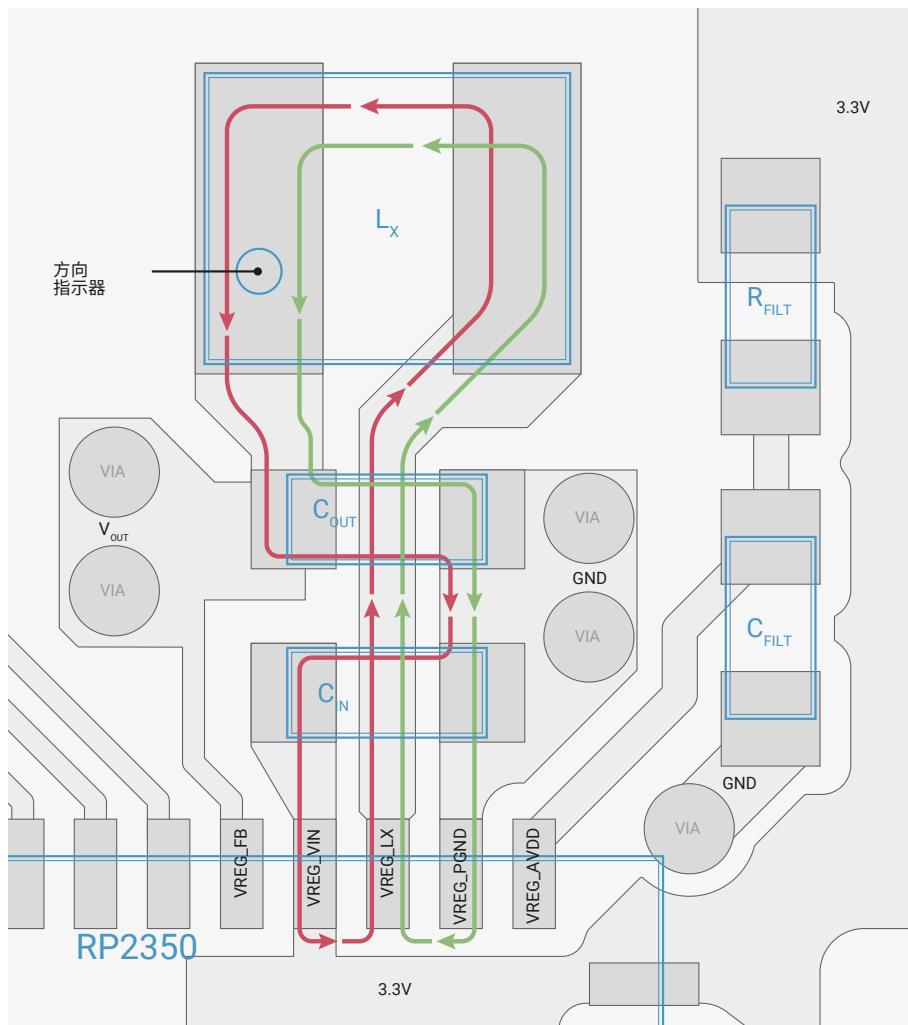


图 23。Raspberry Pi Pico 2 PCB  
布局中的稳压器部分，显示了稳压器各开关相的高电流路径。A  
OTA-B201610S3R3-  
101电感的封装  
尺寸为0806 (201  
6公制)，电阻和电  
容封装为0402 (1  
005公制)



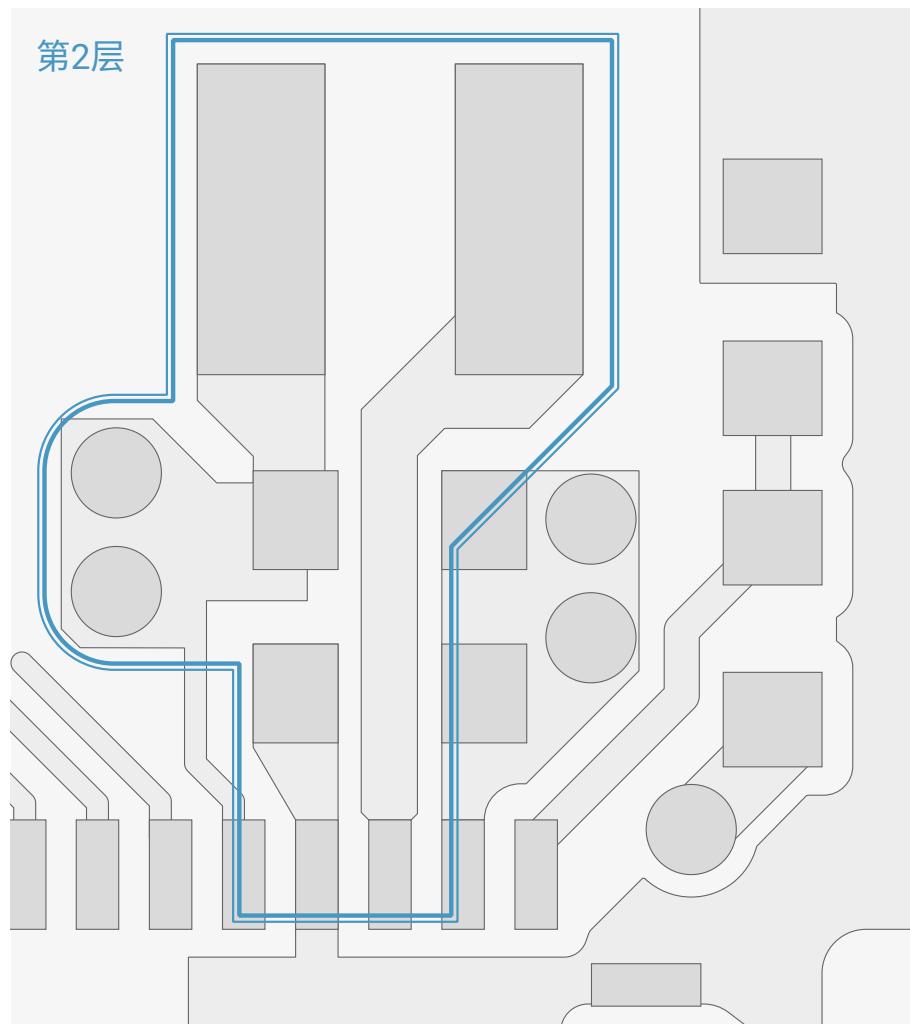
设计人员应尽可能严格遵循上述示意图图22及布局图23，因为这些图经过最充分的验证，且被视为我们的最佳实践布局方案。本电路设计应用于Raspberry Pi Pico 2及RP2350参考设计（详见RP2350硬件设计，极简设计示例），这两种设计分别以Cadence Allegro与Kicad格式提供。图23展示了Raspberry Pi Pico 2 PCB顶层的稳压器布局。稳压器下方的底层为接地平面，连接至QFN封装的GND中心焊盘。

### 6.3.8.1. 布局建议

- VREG\_AVDD 为噪声敏感信号，必须按照图22进行RC滤波。
  - 避免做任何可能将噪声耦合至 VREG\_AVDD 的行为。
  - C<sub>IN</sub> 需设有独立的接地通孔或低阻抗路径，直接连接至RP2350接地焊盘。
- 图23中红色和绿色箭头标示了各稳压器开关阶段的高电流路径。务必保持这些电流路径的回路面积尽可能小且低阻抗，且保持隔离（即仅在单一节点连接至主接地）。
  - 请尽可能严格遵循此布局。
  - 请勿将任何C<sub>IN</sub>/L<sub>x</sub>/C<sub>OUT</sub>元件安置于PCB背面。
- 降低 VREG\_LX 节点上的寄生效应。
- 在顶层，务必切除电感器下方多余铜箔，并尽可能切除靠近 VREG\_LX 走线的铜箔。

- 对于四层及以上多层板，请切除 $L_x/VREG\_LX$ 节点正下方的所有铜箔。例如，图24展示了这一点。
- 接地过孔的位置至关重要。
  - 必须确保从高电流接地点到Raspberry Pi Pico 2 QFN接地焊盘的接地路径尽可能短且低阻抗（采用两个相邻的过孔以降低阻抗）。
  - 滤波电容 $C_{FILT}$ 也必须有低阻抗且尽可能短的回流路径至QFN接地焊盘（不得与高电流输入/输出电容 $C_{IN}/C_{OUT}$ 共用任何接地过孔）。
- $VREG\_FB$ 引脚应从 $C_{OUT}$ 的输出端供电，避免布线直接经过 $L_x$ 下方。
- $C_{OUT}$ 对稳压器性能和电磁干扰控制至关重要，应置于 $VREG\_VIN$ 与 $VREG\_PGND$ 之间，并尽可能靠近引脚。
  - 除 $C_{OUT}$ 外，为获得最佳性能，我们建议在 $V_{OUT}$ 线上再使用一个 $4.7\mu F$ 的电容。位于封装底部边缘（QFN-60封装上的DVDD引脚23）。请勿将此处置于 $L_x/C_{OUT}$ 附近。

图24。四层（或更多层）PCB的第2层 $L_x/VREG\_LX$ 网络下方的开槽



### 6.3.8.2. 元件数值

- $C_{IN}$  应不少于 $4.7\mu F$ ，且最大寄生电阻不得超过 $50m\Omega$ 。
- $C_{OUT}$  必须为 $4.7\mu F \pm 20\%$ ，最大寄生电阻不超过 $250m\Omega$ ，最大电感不超过 $6nH$ 。
- $L_x$ 必须完全屏蔽，电感值为 $3.3\mu H \pm 20\%$ ，最大直流电阻不超过 $250m\Omega$ 。饱和电流应至少为 $1.5A$ 。电感须标注极性（见图25），并按图23所示进行布局。如以下所述，我们推荐AOTA-B201610S3R3-101-T型号。

### 6.3.8.3. 稳压器灵敏度

RP2350 稳压器具有若干灵敏之处：

- **VREG\_AVDD** 电源对噪声极为敏感。
- 效率对电感电流引起的电感衰减非常敏感，因此需选用低衰减的电感。  
以确保最佳运行效果（通常饱和电流越高效果越佳）。
- 即使使用标称完全屏蔽的电感，泄漏磁场通过电感和输出电容（COUT）形成的输出节点**VREG\_LX**回路耦合，仍可能影响稳压器的控制回路及输出电压。磁场方向（因此电感方向）至关重要——电感必须正确安装，才能确保稳压器在较高输出电流和较大负载瞬变下正常工作。这要求电感具有明确的极性标识。

为满足上述要求，Raspberry Pi 与 Abraccon 合作，开发了一款定制的 $2.0 \times 1.6\text{mm}$   $3.3\mu\text{H}$ 带极性标记电感，型号为AOTA-B2 01610S3R3-101-T（见图25）。这些产品将在适当时间普遍发行，但目前请联系 Raspberry Pi 以申请样品或批量生产。

Raspberry Pi 仍在与稳压器 IP 供应商合作，全面验证并鉴定稳压器及定制电感器。

图 25。带方向  
标记的 AOTA-B201  
610S3R3-10  
1-T 电感器，指示  
电流与磁场方向

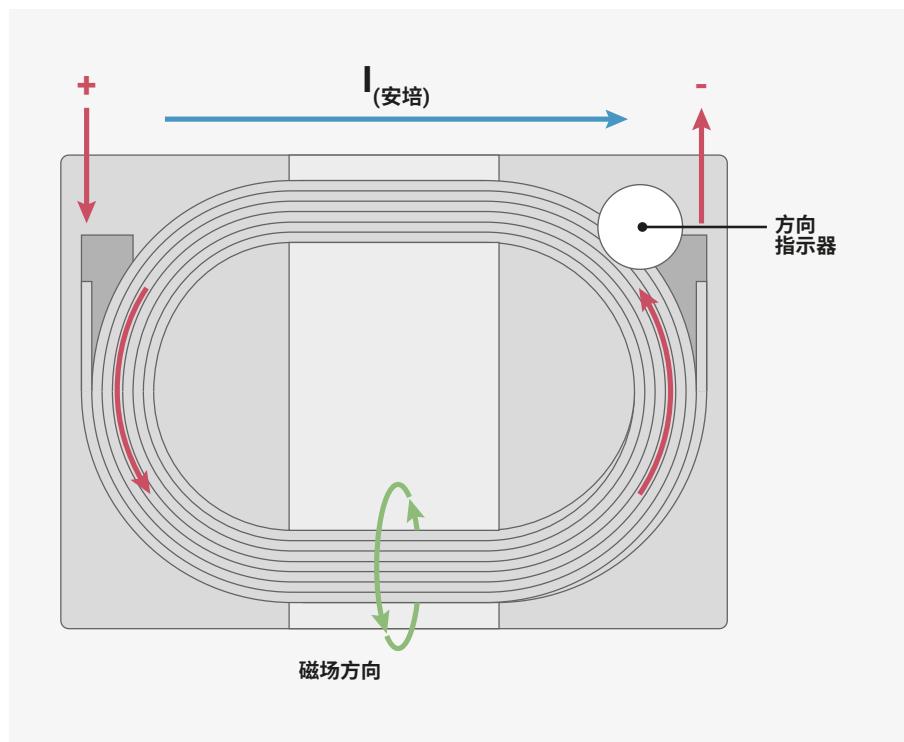
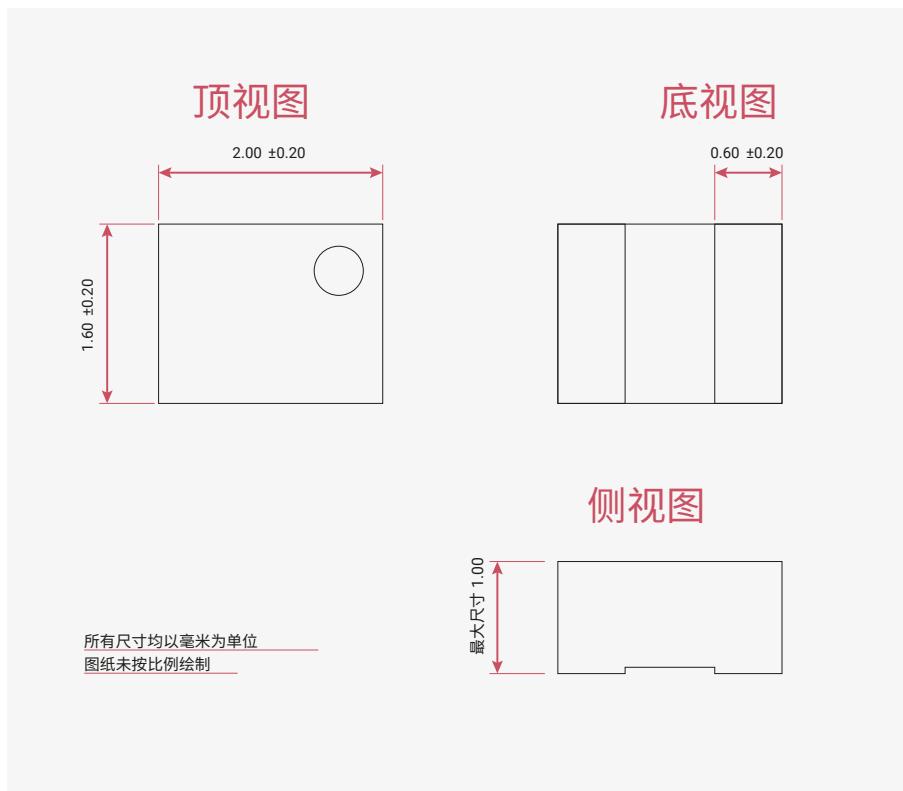


图 26。 AOTA-B2016  
10S3R3-101-  
T 电感器尺寸



### 6.3.9. 寄存器列表

稳压器与始终通电域内其他电源管理子系统共享寄存器地址空间。该地址空间在本文件其他部分称为 **POWMAN**，**POWMAN** 寄存器完整列表详见第 6.4 节。此处重复列出与稳压器相关的 **POWMAN** 寄存器参考信息。

**POWMAN** 寄存器的基址起始于 0x40100000（在 SDK 中定义为 **POWMAN\_BASE**）。

- [VREG\\_CTRL](#)
- [VREG\\_STS](#)
- [VREG](#)
- [VREG\\_LP\\_ENTRY](#)
- [VREG\\_LP\\_EXIT](#)

## 6.4. 电源管理（POWMAN）寄存器

受密码保护的 **POWMAN** 寄存器要求在高 16 位写入密码（0x5AFE）以启用写操作。此措施防止意外写入导致芯片无迹崩溃。对未包含密码的受保护寄存器的写操作将被忽略，并在 **BADPASSWD** 寄存器中置位标志。对受保护寄存器的读取不会返回密码，防止错误的读-改-写操作。

受保护寄存器在高 16 位显然无可写字段，但该范围内可能存在只读字段。

所有地址偏移量最高达且包括 0x000000ac 的寄存器均受密码保护。因此，以下可写寄存器不受保护，支持 32 位写访问：

- POWMAN\_SCRATCH0 → POWMAN\_SCRATCH7
- POWMAN\_BOOT0 → POWMAN\_BOOT3
- POWMAN\_INTR
- POWMAN\_INTE
- POWMAN\_INTF

表477. POWMAN  
寄存器列表

偏移量	名称	说明
0x00	BADPASSWD	表示密码错误
0x04	VREG_CTRL	电压调节器控制
0x08	VREG_STS	电压调节器状态
0x0c	VREG	电压调节器设置
0x10	VREG_LP_ENTRY	电压调节器低功耗进入设置
0x14	VREG_LP_EXIT	电压调节器低功耗退出设置
0x18	BOD_CTRL	欠压检测控制
0x1c	BOD	欠压检测设置
0x20	BOD_LP_ENTRY	欠压检测低功耗进入设置
0x24	BOD_LP_EXIT	欠压检测低功耗退出设置
0x28	LPOSC	低功耗振荡器控制寄存器
0x2c	CHIP_RESET	芯片复位控制与状态
0x30	WDSEL	允许看门狗复位在电源开机状态机（PSM）之外，也复位powman的内部状态。 请注意，powman会忽略未至少选择PSM中CLOCKS阶段或更早阶段的看门狗复位。使用这些位时，建议将PSM_WDSEL设置为全1，并在该寄存器中设置所需位。如果未选择CLOCKS或更早阶段，POWMAN_WDSEL寄存器将不起作用。
0x34	SEQ_CFG	用于配置电源时序器 当POWMAN_STATE_CHANGING=1时，写入操作将被忽略。

偏移量	名称	说明
0x38	状态	<p>该寄存器控制4个电源域的电源状态。 当前电源状态显示在只读的POWMAN_STATE_CURRENT寄存器中。 要更改状态，请写入POWMAN_STATE_REQ寄存器。 POWMAN_STATE_CURRENT和POWMAN_STATE_REQ的编码对应数据手册中定义的电源状态：</p> <p>bit 3 = SWCORE bit 2 = XIP cache bit 1 = SRAM0 bit 0 = SRAM1 0 = 已加电 1 = 已断电 写入 POWMAN_STATE_REQ 后， POWMAN_STATE_WAITING 标志被设置，电源管理器开始确定所需操作。若请求无效转换，电源管理器仍会在 POWMAN_STATE_REQ 中登记该请求，同时设置 POWMAN_BAD_REQ 标志。随后执行加电请求，忽略断电请求。无动作将导致进入不可恢复的死锁状态。无效请求包括：任何加电与断电请求的组合；任何导致 swcore 加电且 xip 断电的请求；若请求断电交换核心域，则 POWMAN_STATE_WAITING 保持激活，直至处理器停止。此期间，POWMAN_STATE_REQ 字段可被重写以更改或取消请求。当电源状态转换开始时，POWMAN_STATE_WAITING 标志将被清除，POWMAN_STATE_CHANGING 标志被置位，且在转换完成前，POWMAN 寄存器的写入操作将被忽略。</p>
0x3c	POW_FASTDIV	
0x40	POW_DELAY	电源状态机延时
0x44	EXT_CTRL0	将 GPIO 配置为电源模式感知的控制输出
0x48	EXT_CTRL1	将 GPIO 配置为电源模式感知的控制输出
0x4c	EXT_TIME_REF	选择一个 GPIO 作为时间参考源，该源可用于驱动 32kHz 的低功耗时钟，或为定时器提供 1ms 的滴答信号，或为定时器提供 1Hz 的滴答信号。滴答信号的选择由 POWMAN_TIMER 寄存器控制。
0x50	LPOSC_FREQ_KHZ_INT	在使用 LPOSC 作为时钟源运行时，通知 AON 定时器时钟频率的整数部分。
0x54	LPOSC_FREQ_KHZ_FRAC	在使用 LPOSC 作为时钟源运行时，通知 AON 定时器时钟频率的小数部分。
0x58	XOSC_FREQ_KHZ_INT	在使用 XOSC 作为时钟源运行时，通知 AON 定时器时钟频率的整数部分。
0x5c	XOSC_FREQ_KHZ_FRAC	当系统由XOSC供电时，向AON定时器报告时钟频率的小数部分。
0x60	SET_TIME_63TO48	
0x64	SET_TIME_47TO32	

偏移量	名称	说明
0x68	SET_TIME_31TO16	
0x6c	SET_TIME_15TO0	
0x70	READ_TIME_UPPER	
0x74	READ_TIME_LOWER	
0x78	ALARM_TIME_63TO48	
0x7c	ALARM_TIME_47TO32	
0x80	ALARM_TIME_31TO16	
0x84	ALARM_TIME_15TO0	
0x88	定时器	
0x8c	PWRUP0	<p>可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。</p> <p>上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。 可用GPIO数量视封装选项而定。 无效选择将被忽略 source = 0 选择 gpio0 . . source = 47 选择 gpio47 source = 48 选择 qspi_ss source = 49 选择 qspi_sd0 source = 50 选择 qspi_sd1 source = 51 选择 qspi_sd2 source = 52 选择 qspi_sd3 source = 53 选择 qspi_sclk level = 0 在 source 为低电平时触发 pwrup level = 1 在 source 为高电平时触发 pwrup</p>
0x90	PWRUP1	<p>可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。</p> <p>上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。 可用GPIO数量视封装选项而定。 无效选择将被忽略 source = 0 选择 gpio0 . . source = 47 选择 gpio47 source = 48 选择 qspi_ss source = 49 选择 qspi_sd0 source = 50 选择 qspi_sd1 source = 51 选择 qspi_sd2 source = 52 选择 qspi_sd3 source = 53 选择 qspi_sclk level = 0 在 source 为低电平时触发 pwrup level = 1 在 source 为高电平时触发 pwrup</p>

偏移量	名称	说明
0x94	PWRUP2	<p>可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。</p> <p>上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。 可用GPIO数量视封装选项而定。 无效选择将被忽略 source = 0 选择 gpio0 · · source = 47 选择 gpio47 source = 48 选择 qspi_ss source = 49 选择 qspi_sd0 source = 50 选择 qspi_sd1 source = 51 选择 qspi_sd2 source = 52 选择 qspi_sd3 source = 53 选择 qspi_sclk level = 0 在 source 为低电平时触发 pwrup level = 1 在 source 为高电平时触发 pwrup</p>
0x98	PWRUP3	<p>可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。</p> <p>上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。 可用GPIO数量视封装选项而定。 无效选择将被忽略 source = 0 选择 gpio0 · · source = 47 选择 gpio47 source = 48 选择 qspi_ss source = 49 选择 qspi_sd0 source = 50 选择 qspi_sd1 source = 51 选择 qspi_sd2 source = 52 选择 qspi_sd3 source = 53 选择 qspi_sclk level = 0 在 source 为低电平时触发 pwrup level = 1 在 source 为高电平时触发 pwrup</p>
0x9c	CURRENT_PWRUP_REQ	<p>指示当前上电请求状态 通过从 pwrup 寄存器中移除使能，可清除 pwrup 事件。通过清除 timer.alarm_enab，可清除警报 pwrup 请求。</p> <p>0 = 芯片复位，有关最近一次复位来源，参见 PO_WMAN_CHIP_RESET。 1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup</p>

偏移量	名称	说明
0xa0	LAST_SWCORE_PWRUP	<p>指示触发最后一次切换核心上电的 pwrup 来源。</p> <p>0 = 芯片复位，有关最近一次复位来源，参见 PO_WMAN_CHIP_RESET。</p> <p>1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup</p>
0xa4	DBG_PWRCFG	
0xa8	BOOTDIS	<p>告知只读存储器忽略下一个 RSM 复位（例如，下一个核心断电 / 上电）后 BOOT0..3 寄存器的内容。</p> <p>如果早期启动阶段已软锁定部分 OTP 页面以保护其内容免受后续阶段访问，存在安全代码在后续阶段通过断电重启核心来解锁这些页面的风险。</p> <p>该寄存器可用于确保引导加载程序在下一次上电时正常运行，从而防止后续阶段的安全代码访问处于未锁定状态的OTP。</p> <p>应与OTP BOOTDIS寄存器配合使用。</p>
0xac	DBGCONFIG	
0xb0	SCRATCH0	临时寄存器。信息可在低功耗模式下保持
0xb4	SCRATCH1	临时寄存器。信息可在低功耗模式下保持
0xb8	SCRATCH2	临时寄存器。信息可在低功耗模式下保持
0xbc	SCRATCH3	临时寄存器。信息可在低功耗模式下保持
0xc0	SCRATCH4	临时寄存器。信息可在低功耗模式下保持
0xc4	SCRATCH5	临时寄存器。信息可在低功耗模式下保持
0xc8	SCRATCH6	临时寄存器。信息可在低功耗模式下保持
0xcc	SCRATCH7	临时寄存器。信息可在低功耗模式下保持
0xd0	BOOT0	临时寄存器。信息可在低功耗模式下保持
0xd4	BOOT1	临时寄存器。信息可在低功耗模式下保持
0xd8	BOOT2	临时寄存器。信息可在低功耗模式下保持
0xdc	BOOT3	临时寄存器。信息可在低功耗模式下保持
0xe0	中断	原始中断
0xe4	INTE	中断使能
0xe8	INTF	中断触发
0xec	INTS	掩码与强制后的中断状态

## POWMAN: BADPASSWD 寄存器

偏移: 0x00

表 478  
BADPASSWD 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	表示密码错误	WC	0x0

## POWMAN: VREG\_CTRL 寄存器

偏移: 0x04

### 描述

电压调节器控制

表 479  
VREG\_CTRL 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>RST_N:</b> 将调节器恢复至启动状态 0 - 复位 1 - 不复位 (默认)	读写	0x1
14	保留。	-	-
13	<b>UNLOCK:</b> 上电后解锁 VREG 控制接口 0 - 锁定 (默认) 1 - 解锁 解锁后不可重新锁定。	读写	0x0
12	<b>ISOLATE:</b> 隔离 VREG 控制接口 0 - 未隔离 (默认) 1 - 已隔离	读写	0x0
11:9	保留。	-	-
8	<b>DISABLE_VOLTAGE_LIMIT:</b> 0=未禁用, 1=已启用	读写	0x0
7	保留。	-	-
6:4	<b>HT_TH:</b> 高温保护阈值 结温超过阈值时, 调节器功率晶体管将被禁用  000 - 100°C 001 - 105°C 010 - 110°C 011 - 115°C 100 - 120°C 101 - 125°C 110 - 135°C 111 - 150°C	读写	0x5
3:2	保留。	-	-
1:0	<b>RESERVED:</b> 写入本字段应置零	读写	0x0

## POWMAN: VREG\_STS 寄存器

偏移: 0x08

### 描述

电压调节器状态

表 480. VREG\_STS 寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>VOUT_OK</b> : 输出调节状态 0=未调节, 1=调节中	只读	0x0
3:1	保留。	-	-
0	<b>STARTUP</b> : 启动状态 0=启动完成, 1=启动中	只读	0x0

## POWMAN: VREG 寄存器

偏移: 0x0c

### 描述

电压调节器设置

表481. VREG 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>UPDATE_IN_PROGRESS</b> : 稳压器状态正在更新 当此字段被设置时, 写入 vreg 寄存器的操作将被忽略	只读	0x0
14:9	保留。	-	-

位	描述	类型	复位
8:4	<b>VSEL</b> : 输出电压选择 稳压器输出电压限制为 1.3V，除非通过 vreg_ctrl 寄存器中的 disable_vo_lstage_limit 字段禁用电压限制 00000 - 0.55V 00001 - 0.60V 00010 - 0.65V 00011 - 0.70V 00100 - 0.75V 00101 - 0.80V 00110 - 0.85V 00111 - 0.90V 01000 - 0.95V 01001 - 1.00V 01010 - 1.05V 01011 - 1.10V (默认) 01100 - 1.15V 01101 - 1.20V 01110 - 1.25V 01111 - 1.30V 10000 - 1.35V 10001 - 1.40V 10010 - 1.50V 10011 - 1.60V 10100 - 1.65V 10101 - 1.70V 10110 - 1.80V 10111 - 1.90V 11000 - 2.00V 11001 - 2.35V 11010 - 2.50V 11011 - 2.65V 11100 - 2.80V 11101 - 3.00V 11110 - 3.15V 11111 - 3.30V	读写	0x0b
3	保留。	-	-
2	<b>RESERVED</b> ：写入本字段应置零	读写	0x0
1	<b>HIZ</b> ：高阻抗模式选择 0 = 非高阻抗模式，1 = 高阻抗模式	读写	0x0
0	保留。	-	-

## POWMAN: VREG\_LP\_ENTRY 寄存器

偏移: 0x10

### 描述

电压调节器低功耗进入设置

表 482  
VREG\_LP\_ENTRY  
寄存器

位	描述	类型	复位
31:9	保留。	-	-

位	描述	类型	复位
8:4	<b>VSEL</b> : 输出电压选择 稳压器输出电压限制为 1.3V，除非通过 vreg_ctrl 寄存器中的 disable_vo_lstage_limit 字段禁用电压限制 00000 - 0.55V 00001 - 0.60V 00010 - 0.65V 00011 - 0.70V 00100 - 0.75V 00101 - 0.80V 00110 - 0.85V 00111 - 0.90V 01000 - 0.95V 01001 - 1.00V 01010 - 1.05V 01011 - 1.10V (默认) 01100 - 1.15V 01101 - 1.20V 01110 - 1.25V 01111 - 1.30V 10000 - 1.35V 10001 - 1.40V 10010 - 1.50V 10011 - 1.60V 10100 - 1.65V 10101 - 1.70V 10110 - 1.80V 10111 - 1.90V 11000 - 2.00V 11001 - 2.35V 11010 - 2.50V 11011 - 2.65V 11100 - 2.80V 11101 - 3.00V 11110 - 3.15V 11111 - 3.30V	读写	0x0b
3	保留。	-	-
2	<b>模式</b> : 选择普通（开关）模式或低功耗（线性）模式 低功耗模式仅适用于输出电压最高为 1.3V 0 = 普通模式（开关） 1 = 低功耗模式（线性）	读写	0x1
1	<b>HIZ</b> : 高阻抗模式选择 0 = 非高阻抗模式，1 = 高阻抗模式	读写	0x0
0	保留。	-	-

## POWMAN: VREG\_LP\_EXIT 寄存器

偏移: 0x14

### 描述

电压调节器低功耗退出设置

表 483。  
VREG\_LP\_EXIT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:4	<b>VSEL</b> ：输出电压选择 稳压器输出电压限制为 1.3V，除非通过 vreg_ctrl 寄存器中的 disable_vo_lstage_limit 字段禁用电压限制 00000 - 0.55V 00001 - 0.60V 00010 - 0.65V 00011 - 0.70V 00100 - 0.75V 00101 - 0.80V 00110 - 0.85V 00111 - 0.90V 01000 - 0.95V 01001 - 1.00V 01010 - 1.05V 01011 - 1.10V (默认) 01100 - 1.15V 01101 - 1.20V 01110 - 1.25V 01111 - 1.30V 10000 - 1.35V 10001 - 1.40V 10010 - 1.50V 10011 - 1.60V 10100 - 1.65V 10101 - 1.70V 10110 - 1.80V 10111 - 1.90V 11000 - 2.00V 11001 - 2.35V 11010 - 2.50V 11011 - 2.65V 11100 - 2.80V 11101 - 3.00V 11110 - 3.15V 11111 - 3.30V	读写	0x0b
3	保留。	-	-
2	模式：选择普通（开关）模式或低功耗（线性）模式 低功耗模式仅适用于输出电压最高为 1.3V 0 = 普通模式（开关） 1 = 低功耗模式（线性）	读写	0x0
1	<b>HIZ</b> ：高阻抗模式选择 0 = 非高阻抗模式，1 = 高阻抗模式	读写	0x0
0	保留。	-	-

## POWMAN: BOD\_CTRL 寄存器

偏移：0x18

### 描述

欠压检测控制

表 484。BOD\_CTRL  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	隔离：隔离欠压检测控制接口 0 - 不隔离（默认） 1 - 已隔离	读写	0x0
11:0	保留。	-	-

## POWMAN：BOD 寄存器

偏移：0x1c

### 说明

欠压检测设置

表 485。BOD  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:4	<b>VSEL</b> ：阈值选择 0000 0 - 0.473V 00001 - 0.516V 00010 - 0.5 59V 00011 - 0.602 V 00100 - 0.645V 0 0101 - 0.688V 001 10 - 0.731V 00111 - 0.774V 01000 - 0. 817V 01001 - 0.86 0V (默认) 01010 - 0.903V 01011 - 0.946 V 01100 - 0.989V 0 1101 - 1.032V 011 10 - 1.075V 01111 - 1.118V  10000 - 1.161 10001 - 1.204V	读写	0x0b
3:1	保留。	-	-
0	<b>EN</b> ：启用褐变检测 0=未启用，1=启用	读写	0x1

## POWMAN：BOD\_LP\_ENTRY 寄存器

偏移：0x20

### 说明

欠压检测低功耗进入设置

表 486。  
BOD\_LP\_ENTRY  
寄存器

位	描述	类型	复位
31:9	保留。	-	-

位	描述	类型	复位
8:4	<b>VSEL:</b> 阈值选择 0000 0 - 0.473V 00001 - 0.516V 00010 - 0.5 59V 00011 - 0.602 V 00100 - 0.645V 0 0101 - 0.688V 001 10 - 0.731V 00111 - 0.774V 01000 - 0. 817V 01001 - 0.86 0V (默认) 01010 - 0.903V 01011 - 0.946 V 01100 - 0.989V 0 1101 - 1.032V 011 10 - 1.075V 01111 - 1.118V  10000 - 1.161 10001 - 1.204V	读写	0x0b
3:1	保留。	-	-
0	<b>EN:</b> 启用褐变检测 0=未启用, 1=启用	读写	0x0

## POWMAN: BOD\_LP\_EXIT 寄存器

偏移: 0x24

### 描述

欠压检测低功耗退出设置

表 487。  
BOD\_LP\_EXIT 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:4	<b>VSEL:</b> 阈值选择 0000 0 - 0.473V 00001 - 0.516V 00010 - 0.5 59V 00011 - 0.602 V 00100 - 0.645V 0 0101 - 0.688V 001 10 - 0.731V 00111 - 0.774V 01000 - 0. 817V 01001 - 0.86 0V (默认) 01010 - 0.903V 01011 - 0.946 V 01100 - 0.989V 0 1101 - 1.032V 011 10 - 1.075V 01111 - 1.118V  10000 - 1.161 10001 - 1.204V	读写	0x0b
3:1	保留。	-	-

位	描述	类型	复位
0	<b>EN</b> : 启用褐变检测 0=未启用, 1=启用	读写	0x1

## POWMAN: LPOS C 寄存器

偏移: 0x28

### 说明

低功耗振荡器控制寄存器

表 488。LPOS C 寄存器

位	描述	类型	复位
31:10	保留。	-	-
9:4	<b>TRIM</b> : 频率微调 - 微调步长通常为复位频率的1%，但可达3%	读写	0x20
3:2	保留。	-	-
1:0	<b>MODE</b> : 该功能已被移除	读写	0x3

## POWMAN: CHIP\_RESET 寄存器

偏移: 0x2c

### 说明

芯片复位控制与状态

表 489。  
CHIP\_RESET 寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<b>HAD_WATCHDOG_RESET_PSM</b> : 最后一次复位为配置为重置上电状态机的看门狗超时 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 timer无 powman无 swcore无 psm有 且不改变电源状态	只读	0x0
27	<b>HAD_HZD_SYS_RESET_REQ</b> : 最后一次复位为来自危险调试器的系统复位 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 timer无 powman无 swcore无 psm有 且不改变电源状态	只读	0x0

位	描述	类型	复位
26	<b>HAD_GLITCH_DETECT:</b> 最后一次复位因电源故障 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 timer无 powman无 swcore无 psm有 且不改变电源状态	只读	0x0
25	<b>HAD_SWCORE_PD:</b> 最后一次复位为切换核心断电 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 timer无 powman无 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
24	<b>HAD_WATCHDOG_RESET_SWCORE:</b> 上次复位为看门狗超时，配置为复位切换核心 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 timer无 powman无 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
23	<b>HAD_WATCHDOG_RESET_POWMAN:</b> 上次复位为看门狗超时，配置为复位电源管理器 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0

位	描述	类型	复位
22	<b>HAD_WATCHDOG_RESET_POWMAN_ASYNC:</b> 上次复位为看门狗超时，配置为异步复位电源管理器 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag无 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
21	<b>HAD_RESCUE:</b> 上次复位为调试器发起的救援复位 复位以下内容： double_tap 标志 无 DP 无 RPAP 无 rescue_flag 无, 设置此标志 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
20	保留。	-	-
19	<b>HAD_DP_RESET_REQ:</b> 上次复位为来自 ARM 调试器的复位请求 复位以下内容： double_tap标志无 DP无 RPAP无 rescue_flag 是 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
18	<b>HAD_RUN_LOW:</b> 上次复位源自 RUN 引脚 复位以下内容： double_tap标志无 DP 是 RPAP 是 rescue_flag 是 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0

位	描述	类型	复位
17	<b>HAD_BOR:</b> 上次复位来自棕色断电检测模块 复位以下内容： double_tap 标志 是 DP 是 RPAP 是 rescue_flag 是 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
16	<b>HAD_POR:</b> 上次复位来自上电复位 复位以下内容： double_tap 标志 是 DP 是 RPAP 是 rescue_flag 是 定时器 是 电源管理器 是 swcore有 psm有 随后启动电源顺序控制器	只读	0x0
15:5	保留。	-	-
4	<b>RESCUE_FLAG:</b> 该标志由 RP-AP 的救援复位设置。 其目的是在启动只读存储器(bootrom)之前停止，以便从启动死锁中恢复。  调试器可在只读存储器停止后附加，并写入不会导致死锁的可运行代码。	WC	0x0
3:1	保留。	-	-
0	<b>DOUBLE_TAP:</b> 此标志由双击 RUN 触发。它指示 bootcode 进入引导加载程序。	读写	0x0

## POWMAN: WDSEL 寄存器

偏移量: 0x30

### 描述

允许看门狗复位在电源开机状态机（PSM）之外，也复位powman的内部状态。

请注意，powman会忽略未至少选择PSM中CLOCKS阶段或更早阶段的看门狗复位。使用这些位时，建议将PSM\_WDSEL设置为全1，并在该寄存器中设置所需位。如果未选择CLOCKS或更早阶段，POWMAN\_WDSEL寄存器将不起作用。

表490. WDSEL  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<b>RESET_PSM:</b> 设置为1时，监视狗复位将执行完整的上电状态机(PSM)序列。 从用户角度看，其效果等同于设置 RSM_WDSEL_PROC_COLD。 从硬件调试角度看，其效果等同于由毛刺检测器触发的复位。	读写	0x0
11:9	保留。	-	-

位	描述	类型	复位
8	<b>RESET_SWCORE</b> : 设置为1时，监视狗复位将复位切换核心电源域并执行完整的上电状态机(PSM)序列。 从用户角度看，其效果等同于设置 RSM_WDSEL_PROC_COLD。 从硬件调试角度看，其效果等同于切换核心电源域的上电复位。	读写	0x0
7:5	保留。	-	-
4	<b>RESET_POWMAN</b> : 若设置为1，监视器复位将恢复powman默认值，重置定时器，重置切换的核心电源域，并执行完整的上电状态机 (PSM) 序列 此项依赖于clk_ref的运行。若该条件可能不成立，请使用reset_powman_async	读写	0x0
3:1	保留。	-	-
0	<b>RESET_POWMAN_ASYNC</b> : 若设置为1，监视器复位将恢复powman默认值，重置定时器，重置切换的核心电源域并执行完整的上电状态机 (PSM) 序列 此项不依赖于clk_ref的运行	读写	0x0

## POWMAN：SEQ\_CFG寄存器

偏移量：0x34

### 描述

用于配置电源时序器

当POWMAN\_STATE\_CHANGING=1时，写入操作将被忽略。

表491. SEQ\_CFG 寄存器

位	描述	类型	复位
31:21	保留。	-	-
20	<b>USING_FAST_POWCK</b> : 0表示POWMAN时钟由低功耗振荡器 (32kHz) 提供 1表示POWMAN时钟由基准时钟 (2-50MHz) 提供	只读	0x1
19:18	保留。	-	-
17	<b>USING_BOD_LP</b> : 表示低功耗欠压检测器 (BOD) 模式 0 = BOD高功率模式，默认为此模式 1 = BOD低功率模式	只读	0x0
16	<b>USING_VREG_LP</b> : 指示电压调节器 (VREG) 工作模式 0 = VREG高功率模式，默认为此模式 1 = VREG低功率模式	只读	0x0
15:13	保留。	-	-
12	<b>USE_FAST_POWCK</b> : 切换核心通电时，选择参考时钟 (clk_ref) 作为POWMAN时钟源切换核心断电时，POWMAN时钟始终切换至慢时钟 (lposc)，因为快速时钟停止运行  0 始终使用慢时钟 (lposc) 运行POWMAN时钟 1 可用时使用快速时钟运行POWMAN时钟 此设置在下一次上电序列执行时生效	读写	0x1
11:9	保留。	-	-

位	描述	类型	复位
8	<b>RUN_LPOSC_IN_LP:</b> 置0时, 切换核心断电将停止低功率振荡器; 若用于驱动定时器, 不建议此设置 此设置在切换核心下一次断电时生效	读写	0x1
7	<b>USE_BOD_HP:</b> 设置为0以防止切换核上电时自动切换到高功耗BOD模式  该设置将于下次切换核上电时生效	读写	0x1
6	<b>USE_BOD_LP:</b> 设置为0以防止切换核断电时自动切换到低功耗BOD模式  此设置在切换核心下一次断电时生效	读写	0x1
5	<b>USE_VREG_HP:</b> 设置为0以防止切换核上电时自动切换到高功耗电压调节器模式  该设置将于下次切换核上电时生效	读写	0x1
4	<b>USE_VREG_LP:</b> 设置为0以防止切换核断电时自动切换到低功耗电压调节器模式  此设置在切换核心下一次断电时生效	读写	0x1
3:2	保留。	-	-
1	<b>HW_PWRUP_SRAM0:</b> 指定切换核从低功耗状态 (P1.xxx) 上电至高功耗状态 (P0.0xx) 时, SRAM0的电源状态。 0=上电 1=保持不变	读写	0x0
0	<b>HW_PWRUP_SRAM1:</b> 指定切换核从低功耗状态 (P1.xxx) 上电至高功耗状态 (P0.0xx) 时, SRAM1的电源状态。 0=上电 1=保持不变	读写	0x0

## POWMAN: 状态寄存器

偏移量: 0x38

### 描述

该寄存器控制4个电源域的电源状态。

当前电源状态显示在只读的POWMAN\_STATE\_CURRENT寄存器中。

要更改状态, 请写入POWMAN\_STATE\_REQ寄存器。

POWMAN\_STATE\_CURRENT和POWMAN\_STATE\_REQ的编码对应数据手册中定义的电源状态:

```

bit 3 = SWCORE
bit 2 = XIP cache
bit 1 = SRAM0
bit 0 = SRAM1
0 = 已加电
1 = 已断电

```

当写入POWMAN\_STATE\_REQ时, POWMAN\_STATE\_WAITING标志被置位, 电源管理器开始确定所需状态。若请求无效转换, 电源管理器仍会在POWMAN\_STATE\_REQ中登记该请求, 同时设置POWMAN\_BAD\_REQ标志。随后执行加电请求, 忽略断电请求。无动作将导致进入不可恢复的死锁状态。无效请求包括: 任何加电与断电请求的组合; 任何导致swcore加电且xip断电的请求; 若请求断电交换核心域, 则POWMAN\_STATE\_WAITING保持激活, 直至处理器停止。在此期间, POWMAN\_STATE\_REQ字段可以被重新写入以更改或取消请求。当电源状态转换开始时, POWMAN\_STATE\_WAITING标志将被清除, POWMAN\_STATE\_CHANGING标志被置位, 且在转换完成前, POWMAN寄存器的写入操作将被忽略。

表 492. 状态寄存器

位	描述	类型	复位
31:14	保留。	-	-
13	<b>CHANGING</b> : 表示电源状态正在变更中	只读	0x0
12	<b>WAITING</b> : 表示电源管理器已接收状态变更请求，且正在等待其他操作完成后执行该请求	只读	0x0
11	<b>BAD_HW_REQ</b> : 无效的硬件发起状态请求，电源开启请求已生效，电源关闭请求被忽略	只读	0x0
10	<b>BAD_SW_REQ</b> : 无效的软件发起状态请求已被忽略	只读	0x0
9	<b>PWRUP WHILE_WAITING</b> : 表示由于已有待处理的电源状态变更请求，新的电源状态变更请求被忽略	WC	0x0
8	<b>REQ IGNORED</b> : 表示软件发起的状态变更请求因与正在进行的硬件或调试器请求冲突而被忽略	WC	0x0
7:4	<b>REQ</b> : 由软件或硬件写入以请求新的电源状态	读写	0x0
3:0	<b>CURRENT</b> : 指示当前电源状态	只读	0xf

## POWMAN: POW\_FASTDIV 寄存器

偏移量: 0x3c

表 493.  
POW\_FASTDIV  
寄存器

位	描述	类型	复位
31:11	保留。	-	-
10:0	对 POWMAN 时钟进行分频，以为延迟模块和状态机提供计时脉冲  当 clk_pow 由慢时钟驱动时，不进行分频 当 clk_pow 由快时钟驱动时，按 tick_div 进行分频	读写	0x040

## POWMAN: POW\_DELAY 寄存器

偏移量: 0x40

### 描述

电源状态机延时

表 494.  
POW\_DELAY  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:8	<b>SRAM_STEP</b> : sram0 与 sram1 电源状态机步骤间的时序间隔  以 powman 计时周期单位测量 ( $\geq 1\mu s$ )，0 表示延迟 1 个单位	读写	0x20
7:4	<b>XIP_STEP</b> : xip 电源状态机步骤间的时序间隔，以 lposc 周期为单位，0 表示延迟 1 个单位	读写	0x1
3:0	<b>SWCORE_STEP</b> : swcore 电源状态机步骤间的时序间隔 以 lposc 周期为单位测量，0 表示延迟 1 个周期	读写	0x1

## POWMAN: EXT\_CTRL0 寄存器

偏移: 0x44

### 说明

将 GPIO 配置为电源模式感知的控制输出

表 495. EXT\_CTRL0  
寄存器

位	描述	类型	复位
31:15	保留。	-	-
14	<b>LP_EXIT_STATE</b> : 退出低功耗状态时的输出电平	读写	0x0
13	<b>LP_ENTRY_STATE</b> : 进入低功耗状态时的输出电平	读写	0x0
12	<b>INIT_STATE</b>	读写	0x0
11:9	保留。	-	-
8	<b>初始化</b>	读写	0x0
7:6	保留。	-	-
5:0	<b>GPIO_SELECT</b> : 选择 gpio 0 → 30 设置为 31 以禁用此功能	读写	0x3f

## POWMAN: EXT\_CTRL1 寄存器

偏移: 0x48

### 说明

将 GPIO 配置为电源模式感知的控制输出

表 496. EXT\_CTRL1  
寄存器

位	描述	类型	复位
31:15	保留。	-	-
14	<b>LP_EXIT_STATE</b> : 退出低功耗状态时的输出电平	读写	0x0
13	<b>LP_ENTRY_STATE</b> : 进入低功耗状态时的输出电平	读写	0x0
12	<b>INIT_STATE</b>	读写	0x0
11:9	保留。	-	-
8	<b>初始化</b>	读写	0x0
7:6	保留。	-	-
5:0	<b>GPIO_SELECT</b> : 选择 gpio 0 → 30 设置为 31 以禁用此功能	读写	0x3f

## POWMAN: EXT\_TIME\_REF 寄存器

偏移: 0x4c

### 说明

选择一个 GPIO 作为时间参考源，该源可用于驱动 32kHz 的低功耗时钟，或为定时器提供 1ms 的滴答信号，或为定时器提供 1Hz 的滴答信号。滴答信号的选择由 POWMAN\_TIMER 寄存器控制。

表 497.  
EXT\_TIME\_REF  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>DRIVE_LPCK</b> : 使用所选 GPIO 驱动 32kHz 低功耗时钟，替代 LPOSOC。该字段仅在 POWMAN_TIMER_RUN=0 时允许写入。	读写	0x0
3:2	保留。	-	-

位	描述	类型	复位
1:0	<b>SOURCE_SEL</b> : 0 → gpio12 1 → gpio20 2 → gpio14 3 → gpio22	读写	0x0
	枚举值:		
	0x0 → GPIO12		
	0x1 → GPIO20		
	0x2 → GPIO14		
	0x3 → GPIO22		

## POWMAN: LPOS\_C\_FREQ\_KHZ\_INT 寄存器

偏移: 0x50

### 说明

在使用 LPOS\_C 作为时钟源运行时，通知 AON 定时器时钟频率的整数部分。

表 498.  
*LPOS\_C\_FREQ\_KHZ\_I*  
NT 寄存器

位	描述	类型	复位
31:6	保留。	-	-
5:0	LPOS_C 或 GPIO 时钟源频率的千赫整数部分。 默认值=32，且仅在 POWMAN_TIMER_RUN=0 或 POWMAN_TIMER_USING_X OSC=1 时允许写入此字段。	读写	0x20

## POWMAN: LPOS\_C\_FREQ\_KHZ\_FRAC 寄存器

偏移: 0x54

### 说明

在使用 LPOS\_C 作为时钟源运行时，通知 AON 定时器时钟频率的小数部分。

表 499.  
*LPOS\_C\_FREQ\_KHZ\_F*  
RAC 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	LPOS_C 或 GPIO 时钟源频率的分数部分，单位为 kHz。 默认值 = 0.768，本字段仅在 POWMAN_TIMER_RUN=0 或 POWMAN_TIMER_U SING_XOSC=1 时允许写入。	读写	0xc49c

## POWMAN: XOSC\_FREQ\_KHZ\_INT 寄存器

偏移: 0x58

### 描述

在使用 XOSC 作为时钟源运行时，通知 AON 定时器时钟频率的整数部分。

表 500。  
XOSC\_FREQ\_KHZ\_INT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	XOSC 频率的整数部分，单位为 kHz。默认值为 12000，且必须大于 1。 本字段仅在 POWMAN_TIMER_RUN=0 或 POWMAN_TIMER_USIN_G_XOSC=0 时允许写入。	读写	0x2ee0

## POWMAN: XOSC\_FREQ\_KHZ\_FRAC 寄存器

偏移: 0x5c

### 描述

当系统由XOSC供电时，向AON定时器报告时钟频率的小数部分。

表 501。  
XOSC\_FREQ\_KHZ\_FRAC  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	XOSC 频率的分数部分，单位为 kHz。本字段仅在 POWMAN_TIMER_RUN=0 或 POWMAN_TIMER_USING_XOSC=0 时允许写入。	读写	0x0000

## POWMAN: SET\_TIME\_63T048 寄存器

偏移: 0x60

表 502  
SET\_TIME\_63T048  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	用于设置时间，禁止用于读取时间，请改用 POWMAN_READ_TIME_UPPER 和 POWMAN_READ_TIME_LOWER。该字段仅在 POWMAN_TIMER_RUN=0 时允许写入。	读写	0x0000

## POWMAN: SET\_TIME\_47T032 寄存器

偏移: 0x64

表 503  
SET\_TIME\_47T032  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	用于设置时间，禁止用于读取时间，请改用 POWMAN_READ_TIME_UPPER 和 POWMAN_READ_TIME_LOWER。该字段仅在 POWMAN_TIMER_RUN=0 时允许写入。	读写	0x0000

## POWMAN: SET\_TIME\_31T016 寄存器

偏移: 0x68

表 504  
SET\_TIME\_31T016  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	用于设置时间，禁止用于读取时间，请改用 POWMAN_READ_TIME_UPPER 和 POWMAN_READ_TIME_LOWER。该字段仅在 POWMAN_TIMER_RUN=0 时允许写入。	读写	0x0000

## POWMAN: SET\_TIME\_15T00 寄存器

偏移: 0x6c

表 505  
SET\_TIME\_15T00  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	用于设置时间，禁止用于读取时间，请改用 POWMAN_READ_TIME_UPPER 和 POWMAN_READ_TIME_LOWER。该字段仅 在 POWMAN_TIMER_RUN=0 时允许写入。	读写	0x0000

## POWMAN：READ\_TIME\_UPPER 寄存器

偏移量: 0x70

表 506  
READ\_TIME\_UPPER  
寄存器

位	描述	类型	复位
31:0	用于读取计时器的位63至32。读取全部64位时，LOWER计数值可能在读取 过程中回绕。建议先读取UPPER，再读取LOWER，随后重新读取UPPER ，若其值发生变化，则应重新读取LOWER。	只读	0x00000000

## POWMAN：READ\_TIME\_LOWER 寄存器

偏移量: 0x74

表 507  
READ\_TIME\_LOWER  
寄存器

位	描述	类型	复位
31:0	用于读取计时器的位31至0。	只读	0x00000000

## POWMAN：ALARM\_TIME\_63T048 寄存器

偏移量: 0x78

表 508  
ALARM\_TIME\_63T048  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	仅当POWMAN_ALARM_ENAB=0时，方可写入此字段。	读写	0x0000

## POWMAN：ALARM\_TIME\_47T032 寄存器

偏移: 0x7c

表 509。  
ALARM\_TIME\_47T032  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	仅当POWMAN_ALARM_ENAB=0时，方可写入此字段。	读写	0x0000

## POWMAN：ALARM\_TIME\_31T016 寄存器

偏移: 0x80

表 510。  
ALARM\_TIME\_31T016  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	仅当POWMAN_ALARM_ENAB=0时，方可写入此字段。	读写	0x0000

## POWMAN：ALARM\_TIME\_15T00 寄存器

偏移: 0x84

表 511。  
ALARM\_TIME\_15T00  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	仅当POWMAN_ALARM_ENAB=0时，方可写入此字段。	读写	0x0000

## POWMAN：TIMER 寄存器

偏移: 0x88

表 512。TIMER  
寄存器

位	描述	类型	复位
31:20	保留。	-	-
19	<b>USING_GPIO_1HZ</b> : 定时器同步至 1Hz GPIO 源	只读	0x0
18	<b>USING_GPIO_1KHZ</b> : 定时器运行于 1kHz GPIO 源	只读	0x0
17	<b>USING_LPOSC</b> : 定时器运行于 LPOSC	只读	0x0
16	<b>USING_XOSC</b> : 定时器运行于 XOSC	只读	0x0
15:14	保留。	-	-
13	<b>USE_GPIO_1HZ</b> : 选择 gpio 作为秒计数器的参考源。 毫秒计数器将继续使用 lposc 或 xosc 作为参考源。	读写	0x0
12:11	保留。	-	-
10	<b>USE_GPIO_1KHZ</b> : 切换至 gpio 作为 1kHz 定时器节拍源。	SC	0x0
9	<b>USE_XOSC</b> : 切换至 xosc 作为 1kHz 定时器节拍源。	SC	0x0
8	<b>USE_LPOSC</b> : 切换至 lposc 作为 1kHz 定时器节拍源。	SC	0x0
7	保留。	-	-
6	<b>ALARM</b> : 闹钟已触发。写入 1 以清除闹钟标志。	WC	0x0
5	<b>PWRUP_ON_ALARM</b> : 闹钟将芯片从低功耗模式唤醒。	读写	0x0
4	<b>ALARM_ENAB</b> : 启用闹钟功能。设置闹钟时间时，必须先禁用闹钟。	读写	0x0
3	保留。	-	-
2	<b>CLEAR</b> : 清除定时器计数，不禁用定时器且不影响闹钟状态。此控制位可随时写入。	SC	0x0
1	<b>RUN</b> : 定时器使能位。设置此位后，定时器将从当前计数值开始递增计数。清除此位将停止计时器计数。  在启用计时器之前，须设置 POWMAN_LPOSC_FREQ* 和 POWMAN_XOSC_FREQ* 寄存器以配置计数速率，并通过写入 SET_TIME_63T048 至 SET_TIME_15T00 寄存器初始化当前时间。计时器运行时，严禁写入 SET_TIME_X 寄存器。  配置完成后，通过设置 POWMAN_TIMER_RUN=1 启动计时器，使计时器自 LPOSC 开始计数。当 XOSC 可用时，将参考时钟切换至 XOSC，随后通过设置 POWMAN_TIMER_USE_XOSC=1 选择其为计时器时钟。	读写	0x0
0	<b>NONSEC_WRITE</b> : 控制非安全软件是否允许写入计时器寄存器。所有其他寄存器均被硬连线设定为非安全不可访问。	读写	0x0

## POWMAN：PWRUPO 寄存器

偏移: 0x8c

**描述**

可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。

上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。

GPIO 的数量取决于封装选项。无效选择将被忽略

source = 0 选择 gpio0

1. +

2. + source = 47 选择 gpio47

source = 48 选择 qspi\_ss

source = 49 选择 qspi\_sd0

source = 50 选择 qspi\_sd1

source = 51 选择 qspi\_sd2

source = 52 选择 qspi\_sd3

source = 53 选择 qspi\_sclk

level = 0 在 source 为低电平时触发 pwrup

level = 1 在 source 为高电平时触发 pwrup

表513. PWRUP0 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>RAW_STATUS</b> : 所选 gpio 引脚的数值（仅当 enable == 1 时有效）	只读	0x0
9	<b>STATUS</b> : gpio 唤醒状态。写入 1 即可清除锁存的边缘检测。	WC	0x0
8	<b>MODE</b> : 边缘检测或电平检测。边缘检测会检测从 0 到 1 (或 1 到 0) 的跳变。电平检测会检测逻辑 1 或逻辑 0。两种事件类型均锁存至 current_pwrup_req 寄存器。	读写	0x0
	枚举值:		
	0x0 → 电平 (LEVEL)		
	0x1 → 边缘 (EDGE)		
7	<b>方向</b>	读写	0x0
	枚举值:		
	0x0 → 低电平下降 (LOW_FALLING)		
	0x1 → 高电平上升 (HIGH_RISING)		
6	<b>ENABLE</b> : 置 1 以启用唤醒源。置 0 以禁用唤醒源并清除待处理唤醒事件。  使用边缘检测时，须向状态寄存器写入 1 以清除锁存的边缘。	读写	0x0
5:0	<b>来源</b>	读写	0x3f

**POWMAN: PWRUP1 寄存器**

偏移: 0x90

**描述**

可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。

上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。

GPIO 的数量取决于封装选项。无效选择将被忽略

source = 0 选择 gpio0

1. +

2. + source = 47 选择 gpio47  
 source = 48 选择 qspi\_ss  
 source = 49 选择 qspi\_sd0  
 source = 50 选择 qspi\_sd1  
 source = 51 选择 qspi\_sd2  
 source = 52 选择 qspi\_sd3  
 source = 53 选择 qspi\_sclk  
 level = 0 在 source 为低电平时触发 pwrup  
 level = 1 在 source 为高电平时触发 pwrup

表514。PWRUP1 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>RAW_STATUS</b> : 所选 gpio 引脚的数值（仅当 enable == 1 时有效）	只读	0x0
9	<b>STATUS</b> : gpio 唤醒状态。写入 1 即可清除锁存的边缘检测。	WC	0x0
8	<b>MODE</b> : 边缘检测或电平检测。边缘检测会检测从 0 到 1 (或 1 到 0) 的跳变。电平检测会检测逻辑 1 或逻辑 0。两种事件类型均锁存至 current_pwrup_req 寄存器。	读写	0x0
	枚举值：		
	0x0 → 电平 (LEVEL)		
	0x1 → 边缘 (EDGE)		
7	<b>方向</b>	读写	0x0
	枚举值：		
	0x0 → 低电平下降 (LOW_FALLING)		
	0x1 → 高电平上升 (HIGH_RISING)		
6	<b>ENABLE</b> : 置 1 以启用唤醒源。置 0 以禁用唤醒源并清除待处理唤醒事件。  使用边缘检测时，须向状态寄存器写入 1 以清除锁存的边缘。	读写	0x0
5:0	<b>来源</b>	读写	0x3f

## POWMAN：PWRUP2 寄存器

偏移: 0x94

### 描述

可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。  
 上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。  
 GPIO 的数量取决于封装选项。无效选择将被忽略  
 source = 0 选择 gpio0

1. +

2. + source = 47 选择 gpio47  
 source = 48 选择 qspi\_ss  
 source = 49 选择 qspi\_sd0  
 source = 50 选择 qspi\_sd1  
 source = 51 选择 qspi\_sd2  
 source = 52 选择 qspi\_sd3  
 source = 53 选择 qspi\_sclk  
 level = 0 在 source 为低电平时触发 pwrup

level = 1 在 source 为高电平时触发 pwrup

表515。PWRUP2 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>RAW_STATUS</b> : 所选 gpio 引脚的数值（仅当 enable == 1 时有效）	只读	0x0
9	<b>STATUS</b> : gpio 唤醒状态。写入 1 即可清除锁存的边缘检测。	WC	0x0
8	<b>MODE</b> : 边缘检测或电平检测。边缘检测会检测从 0 到 1（或 1 到 0）的跳变。电平检测会检测逻辑 1 或逻辑 0。两种事件类型均锁存至 current_pwrup_req 寄存器。	读写	0x0
	枚举值：		
	0x0 → 电平 (LEVEL)		
	0x1 → 边缘 (EDGE)		
7	<b>方向</b>	读写	0x0
	枚举值：		
	0x0 → 低电平下降 (LOW_FALLING)		
	0x1 → 高电平上升 (HIGH_RISING)		
6	<b>ENABLE</b> : 置 1 以启用唤醒源。置 0 以禁用唤醒源并清除待处理唤醒事件。 使用边缘检测时，须向状态寄存器写入 1 以清除锁存的边缘。	读写	0x0
5:0	<b>来源</b>	读写	0x3f

## POWMAN: PWRUP3 寄存器

偏移: 0x98

### 描述

可配置4个GPIO上电事件，用于从低功耗状态唤醒芯片。

上电事件对电平或边缘敏感，可设置为在高电平/上升沿或低电平/下降沿时触发。

GPIO 的数量取决于封装选项。无效选择将被忽略

source = 0 选择 gpio0

1. +

2. + source = 47 选择 gpio47

source = 48 选择 qspi\_ss

source = 49 选择 qspi\_sd0

source = 50 选择 qspi\_sd1

source = 51 选择 qspi\_sd2

source = 52 选择 qspi\_sd3

source = 53 选择 qspi\_sclk

level = 0 在 source 为低电平时触发 pwrup

level = 1 在 source 为高电平时触发 pwrup

表516。PWRUP3 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>RAW_STATUS</b> : 所选 gpio 引脚的数值（仅当 enable == 1 时有效）	只读	0x0
9	<b>STATUS</b> : gpio 唤醒状态。写入 1 即可清除锁存的边缘检测。	WC	0x0

位	描述	类型	复位
8	<b>MODE:</b> 边缘检测或电平检测。边缘检测会检测从 0 到 1 (或 1 到 0) 的跳变。电平检测会检测逻辑 1 或逻辑 0。两种事件类型均锁存至 current_pwr_up_req 寄存器。	读写	0x0
	枚举值：		
	0x0 → 电平 (LEVEL)		
	0x1 → 边缘 (EDGE)		
7	<b>方向</b>	读写	0x0
	枚举值：		
	0x0 → 低电平下降 (LOW_FALLING)		
	0x1 → 高电平上升 (HIGH_RISING)		
6	<b>ENABLE:</b> 置 1 以启用唤醒源。置 0 以禁用唤醒源并清除待处理唤醒事件。 使用边缘检测时，须向状态寄存器写入 1 以清除锁存的边缘。	读写	0x0
5:0	<b>来源</b>	读写	0x3f

## POWMAN: CURRENT\_PWRUP\_REQ 寄存器

偏移: 0x9c

表517。  
*CURRENT\_PWRUP\_R*  
*EQ* 寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:0	指示当前上电请求状态 通过从 pwrup 寄存器中移除使能，可以清除 pwrup 事件。 通过清除 timer.alarm_enab，可清除警报 pwrup 请求。 0 = 芯片复位，有关最近一次复位来源，参见 POWMAN_CHIP_RESET。 1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup	只读	0x00

## POWMAN: LAST\_SWCORE\_PWRUP 寄存器

偏移: 0xa0

表518。  
LAST\_SWCORE\_PWRU  
P 寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:0	指示触发最后一次切换核心上电的电源来源 0 = 芯片复位，有关最近一次复位来源，参见 POWMAN_CHIP_RESET。 1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup	只读	0x00

## POWMAN: DBG\_PWRCFG 寄存器

偏移: 0xa4

表519。  
DBG\_PWRCFG  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>IGNORE:</b> 忽略调试器的上电请求。若上电请求被断言，此项将阻止断电并设置断电阻止状态。设置忽略以停止响应上电请求	读写	0x0

## POWMAN: BOOTDIS 寄存器

偏移: 0xa8

### 描述

告知只读存储器忽略下一个 RSM 复位（例如，下一个核心断电/上电）后 BOOT0..3 寄存器的内容。

如果早期启动阶段已软锁定部分 OTP 页面以保护其内容免受后续阶段访问，存在安全代码在后续阶段通过断电重启核心来解锁这些页面的风险。

该寄存器可用于确保引导加载程序在下一次上电时正常运行，从而防止后续阶段的安全代码访问处于未锁定状态的OTP。

应与OTP BOOTDIS寄存器配合使用。

表520. BOOTDIS  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>NEXT:</b> 该标志写入操作总是与其当前内容做按位或（OR）操作。软件可以设置该位，但不能清除此位。  当核心断电时，BOOTDIS_NEXT 位会按位或（OR）进 BOOTDIS_NOW 位。 同时，BOOTDIS_NEXT 位会被清零。设置此位表示，在powman对 RSM 进行下一次复位后，BOOT0..3 寄存器将被忽略。  此标志应由具备软锁OTP页面功能的早期引导阶段设置，以防止后续阶段通过断电重启进行解锁。	读写	0x0

位	描述	类型	复位
0	<b>NOW:</b> 当powman重置RSM时，BOOTDIS_NEXT的当前值将被按位或至BOOTDIS_NOW，且BOOTDIS_NEXT随即清零。  bootrom在读取BOOT0至BOOT3寄存器前，会检查此标志。若标志被设置，bootrom将其清除并忽略BOOT寄存器内容。此举防止安全软件在引导加载程序软锁包含敏感数据的OTP页面之前篡改引导路径。	WC	0x0

## POWMAN：DBGCONFIG寄存器

偏移: 0xac

表 521。DBGCONFIG 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	<b>DP_INSTID:</b> 配置SWD多路选择的DP实例ID。 建议仅在多芯片环境需要调试访问时修改此配置。	读写	0x0

## POWMAN：SCRATCH0、SCRATCH1、...、SCRATCH6、SCRATCH7寄存器

偏移量: 0xb0, 0xb4, ..., 0xc8, 0xcc

表 522。SCRATCH0、SCRATCH1、...、SCRATCH6、SCRATCH7 寄存器

位	描述	类型	复位
31:0	临时寄存器。信息可在低功耗模式下保持	读写	0x00000000

## POWMAN：BOOT0、BOOT1、BOOT2、BOOT3 寄存器

偏移量: 0xd0, 0xd4, 0xd8, 0xdc

表 523。BOOT0、BOOT1、BOOT2、  
BOOT3 寄存器

位	描述	类型	复位
31:0	临时寄存器。信息可在低功耗模式下保持	读写	0x00000000

## POWMAN：INTR 寄存器

偏移量: 0xe0

### 描述

原始中断

表 524。INTR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>PWRUP WHILE_WAITING:</b> 来源于 state.pwrup_while_waiting	只读	0x0
2	<b>STATE_REQ_IGNORED:</b> 来源于 state.req_ignored	只读	0x0
1	<b>定时器</b>	只读	0x0
0	<b>VREG_OUTPUT_LOW</b>	WC	0x0

## POWMAN：INTE 寄存器

偏移量: 0xe4

**描述**

中断使能

表 525。INTE 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>PWRUP WHILE_WAITING</b> : 来源于 state.pwrup_while_waiting	读写	0x0
2	<b>STATE_REQ_IGNORED</b> : 来源于 state.req_ignored	读写	0x0
1	<b>定时器</b>	读写	0x0
0	<b>VREG_OUTPUT_LOW</b>	读写	0x0

**POWMAN: INTF 寄存器**

偏移: 0xe8

**描述**

中断强制

表 526。INTF 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>PWRUP WHILE_WAITING</b> : 来源于 state.pwrup_while_waiting	读写	0x0
2	<b>STATE_REQ_IGNORED</b> : 来源于 state.req_ignored	读写	0x0
1	<b>定时器</b>	读写	0x0
0	<b>VREG_OUTPUT_LOW</b>	读写	0x0

**POWMAN: INTS 寄存器**

偏移: 0xec

**描述**

掩码与强制后的中断状态

表 527。INTS 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>PWRUP WHILE_WAITING</b> : 来源于 state.pwrup_while_waiting	只读	0x0
2	<b>STATE_REQ_IGNORED</b> : 来源于 state.req_ignored	只读	0x0
1	<b>定时器</b>	只读	0x0
0	<b>VREG_OUTPUT_LOW</b>	只读	0x0

**6.5. 电源降低策略**

RP2350 保留了 RP2040 用于动态功耗控制的睡眠 (SLEEP) 和休眠 (DORMANT) 状态，并通过引入电源域（第 6.2.1 节）对其进行了扩展，允许对芯片上的各个组件断电，几乎消除泄漏电流，从而支持更低功耗模式。

### 6.5.1. 顶层时钟门控

每个时钟域（例如系统时钟）可能驱动大量不同的硬件模块，且并非所有模块都需要同时启用。为避免不必要的功耗，每个时钟的各个端点（例如 UART 系统时钟输入）均可随时被禁用。

时钟门控的启用和禁用过程无毛刺现象。若外设时钟被临时禁用，随后重新启用，外设将恢复到禁用前的状态。不应需要复位或重新初始化。

时钟门控由两组寄存器控制：[WAKE\\_ENx](#)寄存器（从 WAKE\_EN0 开始）和 [SLEEP\\_ENx](#)寄存器（从 SLEEP\_EN0 开始）。这两组寄存器在位级上完全相同，均包含用于控制各时钟端点的标志位。[WAKE\\_EN](#)寄存器指定系统处于唤醒状态时启用的时钟，[SLEEP\\_ENx](#)寄存器则选择处理器处于 [SLEEP](#) 状态时启用的时钟（参见第6.5.2节）。

两个处理器均无外部可控的时钟门控功能。处理器基于执行 [WFI](#)/[WFE](#)指令以及外部事件和中断信号，自主对其子系统的时钟进行门控。

### 6.5.2. 睡眠状态

当满足下列所有条件时，RP2350 进入 SLEEP 状态：

- 两个处理器均处于睡眠状态（例如执行 [WFE](#) 或 [WFI](#) 指令时）
- 系统DMA在任何通道上均无未完成的传输。

当任一处理器被中断唤醒时，RP2350退出SLEEP状态。

在SLEEP状态下，顶层时钟门控由 [SLEEP\\_ENx](#)寄存器（从SLEEP\_EN0起）控制掩码，而非 [WAKE\\_ENx](#)寄存器（从WAKE\_EN0起）。这使得在处理器休眠时能够更积极地裁剪时钟树。

#### 注意

尽管时钟可以在SLEEP期间启用而在非SLEEP期间禁用，但这通常无实际意义。

例如，若系统休眠直至UART字符中断，则除UART外，整个系统均可进行时钟门控控制（[SLEEP\\_ENx](#) = 全部为零，唯独 [CLK\\_SYS\\_UART0](#) 和 [CLK\\_PERI\\_UART0](#)例外）。这包括诸如总线结构等系统基础设施。

当UART触发中断并唤醒处理器时，RP2350退出SLEEP模式并切换回[WAKE\\_ENx](#)时钟掩码。至少应包括总线结构和包含处理器堆栈及中断向量的存储器设备。

系统级时钟请求握手将在时钟重新启用之前，令处理器停用总线。

### 6.5.3. 休眠状态

休眠状态（DORMANT）为真正的零动态功耗睡眠状态，其中所有时钟（包括所有振荡器）均被禁用。系统可通过GPIO事件（高/低电平或上升/下降沿）或AON定时器警报唤醒：这将重新启动一个振荡器（环振荡器或晶体振荡器），并在振荡器稳定后开启振荡器输出门控。系统状态得以保留，退出休眠状态（DORMANT）后代码执行立即恢复。

若依赖AON定时器（第12.10节）从休眠状态唤醒，AON定时器必须由LPOSC或外部时钟源驱动。AON定时器支持低至1 Hz的时钟频率。

DORMANT 状态不会停止锁相环（PLL）。为避免不必要的功耗，软件应在进入 DORMANT 状态前关闭 PLL，退出后重新开启并重新配置 PLL。

如果停止晶振（XOSC），必须同时停止PLL，以防其输入参考时钟停止时PLL失锁。当频率参考丢失时，PLL的电压控制振荡器（VCO）可能表现异常，例如频率升至异常高值。XOSC重新启动后，应重新配置并启用PLL。XOSC停止期间，不得尝试由PLL供时钟运行。

DORMANT状态通过向当前活动的任一振荡器（环振荡器（见第8.3节）或晶振（见第8.2节））的DORMANT寄存器写入关键字来进入。若两者均处于活动状态，必须最后停止为处理器供时的振荡器，因为其停止将阻止软件执行。

### 6.5.3.1. 从 DORMANT 状态唤醒

系统在以下任一事件发生时退出休眠（DORMANT）状态：

- 来自AON计时器的报警，导致TIMER.ALARM被置位
- 来自GPIO银行0的中断被断言至DORMANT\_WAKE中断目标
- 来自GPIO银行1的中断被断言至DORMANT\_WAKE中断目标

从AON计时器唤醒时，**无需启用POWMAN的IRQ输出。计时器触发即可，无需映射至中断输出。任何导致TIMER.ALARM置位的AON计时器报警比较事件，均会使系统退出休眠状态。真正导致退出的是报警事件本身，而非TIMER.ALARM状态；若进入休眠状态时TIMER.ALARM状态为1，但计时器报警比较逻辑被TIMER.ALARM\_ENAB禁用，系统将不会退出休眠状态。**

GPIO银行寄存器设有针对休眠模式唤醒逻辑的中断使能寄存器，如DORMANT\_WAKE\_INTE0。其与针对处理器的中断使能寄存器（如PROCO\_INTE0）完全相同。

从DORMANT（休眠）状态唤醒将重新启动因进入DORMANT状态而被禁用的振荡器。此操作不会重新启动任何其他振荡器，也不会更改系统级时钟配置。

### 6.5.4. 内存外围断电

主系统内存（SRAM0至SRAM9，映射至总线地址0x20000000至0x20081fff）及USB DPRAM，均可通过SYSCFG寄存器中的MEMPOWERDOWN寄存器实现部分断电（详见第12.15.2节）。该操作关闭了用于访问SRAM存储阵列的模拟电路（即SRAM的外围电路），但存储阵列本身保持供电。内存内容保持不变，但无法访问。静态功耗有所降低。

#### ⚠ 警告

内存断电状态下严禁访问。否则可能导致内存内容损坏。

对内存重新供电后，必须延迟至少20纳秒后方可访问内存。

XIP缓存（详见第4.4节）亦可通过CTRL.POWER\_DOWN断电。缓存断电期间，XIP硬件不会生成缓存访问请求。请注意，由于外部QSPI总线的较高电压和开关电容，如果代码继续从XIP执行，通常不会带来净功率节约。

### 6.5.5. 全内存断电

RP2350能够完全关闭其内部SRAM。与第6.5.4节中描述的存储外围电源关闭不同，此操作完全断开SRAM电源，使静态功率降至近乎零。

存储器在完全断电时，其内容将全部丢失。当您在断电后重新为存储器供电时，

其内容是完全未定义的。

SRAM具有三个独立的电源域：

#### SRAM0

包含主系统SRAM，地址范围为 `0x20000000` 至 `0x2003ffff` (SRAM银行0至3)。

#### SRAM1

包含主系统SRAM，地址范围为 `0x20040000` 至 `0x20081fff` (SRAM银行4至9)。

#### XIP

包含XIP缓存及启动RAM。

当切换核心域通电时，XIP电源域始终保持供电。切换核心域指包括所有核心逻辑的域，如处理器、总线结构和外设。这意味着该域内的存储器在软件运行时始终保持通电。

除了关闭存储器以节省功耗外，还可以在关闭切换核心域的同时保持存储器通电。此举在保留SRAM中程序状态的同时，消除了核心逻辑的静态功耗。

更多信息请参见：

- [第4章 — RP2350存储资源列表，包括主系统SRAM、XIP缓存和启动RAM](#)
- [第6.2.1节 — 核心电源域定义，包括上述列举的存储电源域](#)
- [第6.2.2节 — 支持的存储电源状态列表](#)
- [第6.2.3节 — 启动存储电源状态切换以实现存储通电或断电的信息](#)
- [第14.9.7.2节 — 包括存储断电在内的典型低功耗状态功耗](#)

## 6.5.6. 程序员模型

### 6.5.6.1 睡眠

`hello_sleep`示例（位于[pico-playground GitHub仓库](#)的`hello_sleep_aon.c`中）演示了睡眠模式。`hello_sleep`应用程序（及其底层函数）执行以下步骤：

1. 将系统中所有时钟切换为由XOSC驱动。
2. 在AON定时器中设置一个10秒后的闹钟。
3. 使用`SLEEP_ENx`寄存器（参见`SLEEP_EN0`）将AON定时器时钟设置为睡眠模式下唯一运行的时钟。
4. 启用处理器的深度睡眠。
5. 调用处理器上的`_wfi`，处理器将进入深度睡眠，直至被AON定时器中断唤醒。
6. 10秒后，AON定时器中断清除闹钟，并调用用户提供的回调函数。
7. 该回调函数结束示例应用程序的执行。

#### 注意

要进入睡眠模式，必须在proc0和proc1上启用深度睡眠，调用`_wfi`，并确保DMA已停止。

`hello_sleep`调用了Pico Extras中`pico_sleep`的函数。尤其是，`sleep_goto_sleep_until`使处理器进入睡眠状态，直到被假定在未来的AON定时器时间唤醒。

Pico Extras: [https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\\_common/pico\\_sleep/sleep.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/pico_sleep/sleep.c) 第159至183行

```

159 void sleep_goto_sleep_until(struct timespec *ts, aon_timer_alarm_handler_t callback)
160 {
161
162 // 我们应已调用 sleep_run_from_dormant_source 函数
163 // 该函数仅在休眠模式下需要，尽管它节省了休眠时使用 xosc 运行的功耗
164
165 //assert(dormant_source_valid(_dormant_source));
166
167 clocks_hw->sleep_en0 = CLOCKS_SLEEP_EN0_CLK_REF_POWMAN_BITS;
168 clocks_hw->sleep_en1 = 0x0;
169
170 aon_timer_enable_alarm(ts, callback, false);
171
172 stdio_flush();
173
174 // 启用处理器深度睡眠
175 processor_deep_sleep();
176
177 // 进入睡眠状态
178 __wfi();
179 }
```

### 6.5.6.2. 休眠状态

pico-playground GitHub 仓库中的 `hello_dormant` 示例 (`hello_dormant_gpio.c`) 演示了 DORMANT 状态。该示例执行以下步骤：

1. 将系统中所有时钟切换为由XOSC驱动。
2. 为 `dormant_wake` 硬件配置 GPIO 中断，该硬件可以唤醒处于休眠状态的 ROSC 和 XOSC。
3. 将 XOSC 置于休眠模式，立即停止所有处理器执行及芯片上的所有其他时钟逻辑。
4. 当 GPIO 10 变为高电平时，XOSC 重新启动，程序执行继续。

`hello_dormant` 底层使用了 `sleep_goto_dormant_until_pin`:

Pico 补充资料: [https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\\_common/pico\\_sleep/sleep.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/pico_sleep/sleep.c) 第 258 至 282 行

```

258 void sleep_goto_dormant_until_pin(uint gpio_pin, bool edge, bool high) {
259 bool low = !high;
260 bool level = !edge;
261
262 // 在 IO bank 0 配置相应的中断请求 (IRQ)
263 assert(gpio_pin < NUM_BANK0_GPIOS);
264
265 uint32_t event = 0;
266
267 if (level && low) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_LEVEL_LOW_BITS;
268 if (level && high) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_LEVEL_HIGH_BITS;
269 if (edge && high) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_EDGE_HIGH_BITS;
270 if (edge && low) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_EDGE_LOW_BITS;
271
272 gpio_init(gpio_pin);
273 gpio_set_input_enabled(gpio_pin, true);
274 gpio_set_dormant_irq_enabled(gpio_pin, event, true);
275 }
```

```
276 _go_dormant();
277 // 执行在此处停止，直到被唤醒
278
279 // 清除中断，以便我们可以根据需要重新进入休眠模式
280 gpio_acknowledge_irq(gpio_pin, event);
281 gpio_set_input_enabled(gpio_pin, false);
282 }
```

# 第7章 重置

## 7.1. 概述

重置分为三种类别，分别适用于RP2350的不同子集：

### 芯片级重置

适用于整个芯片。用于将整个芯片恢复到默认状态。此类重置由硬件事件、看门狗或调试器触发。当所有芯片级重置解除断言后，系统重置被释放，处理器开始启动。

### 系统重置

适用于对处理器操作至关重要的组件。系统组件之间存在相互依赖性，因此它们的重置由上电状态机（PSM）按顺序解除断言。完整的PSM序列由芯片级复位的取消断言触发。完整或部分序列可以由看门狗或调试器触发。该序列以处理器启动为终点。

### 子系统复位

适用于非处理器正常运行所必需的组件。这些复位可通过写入RESETS寄存器独立断言，并由软件、看门狗或调试器取消断言。

看门狗可编程以触发上述任意类别。

## 7.2. 与 RP2040 的变更

RP2350保留所有RP2040芯片级复位功能。

RP2350新增以下功能：

- 新的芯片复位源：
  - 毛刺检测器
  - 看门狗
  - 调试器
- 新的复位目标：
  - 新的电源管理组件

RP2350对现有功能作出以下修改：

- 修改了CHIP\_RESET寄存器，该寄存器记录最后一次芯片级复位的来源。在RP2040中，CHIP\_RESET存储于LDO\_POR寄存器块内。在RP2350中，CHIP\_RESET被扩展并移至POWMAN寄存器块，该寄存器块位于新的始终开启电源域（AON）中。
- 将棕断电复位（BOR）寄存器重命名为棕断电检测（BOD），增加了功能，并移至新的POWMAN寄存器块。
- 增加了更多的系统复位阶段。为支持该功能，新增了电源上电状态机字段，并重新排列了现有字段。
- 新增了更多RESETS寄存器，并重新排列了现有字段。
- 扩展了看门狗选项，以支持新的复位触发。

### ⓘ 注意

当看门狗触发芯片级复位时，看门狗临时寄存器不会被保留。然而，在系统或子系统复位后，看门狗临时寄存器会被保留。有关不会在芯片级复位后复位的一般用途临时寄存器，请参阅第6.4节“电源管理（POWMAN）寄存器”中POWMAN寄存器块的说明。

## 7.3. 芯片级复位

芯片级复位会将整个芯片恢复到默认状态。这些复位仅由硬件事件、调试器或看门狗超时触发。

### 7.3.1. 芯片级复位表

表528，“芯片级复位原因列表”展示了各芯片级复位源所复位的组件。破折号（–）表示该源未引起任何变化。

表528。芯片级复位原因列表

复位源	SW-DP	AON 暂存区	POWMAN	电源状态	快速双击	救援
POR	复位	复位	硬件复位	→ P0.0	复位	复位
BOR	复位	复位	硬件复位	→ P0.0	复位	复位
外部复位（运行）	复位	复位	硬件复位	→ P0.0	–	复位
调试器复位请求	–	–	硬件复位	→ P0.0	–	复位
调试器救援	–	–	硬件复位	→ P0.0	–	已设置
看门狗电源管理异步复位	–	–	硬件复位	→ P0.0	–	–
看门狗 POWMAN 复位	–	–	软复位	→ P0.0	–	–
看门狗 SWCORE 复位	–	–	–	→ P0.0	–	–
SWCORE 断电	–	–	–	→ P0.x	–	–
GLITCH_DETECTOR	–	–	–	–	–	–
看门狗复位 PSM	–	–	–	–	–	–

表中所有芯片级复位源亦会复位上电状态机（PSM）。此操作会断言所有下游的系统复位信号。系统复位包括系统级时钟发生器等低层芯片基础设施，以及处理器的冷复位和热复位域。

表中所有芯片级复位源亦会复位系统看门狗外围设备，包括看门狗擦写寄存器 SCRATCH0 → SCRATCH7。

您可按以下方式解读表格列：

#### 复位源

指示“芯片级复位源”中所列事件中，哪个事件导致该芯片级复位。

#### SW-DP

指示软件调试端口（SWD）和 RP-AP（第 3.5.10 节，“RP-AP”）被复位。

#### AON 暂存区

指示 POWMAN 寄存器 SCRATCH0 → SCRATCH7 和 BOOT0 → BOOT3 的擦写寄存器状态已丢失。

这些寄存器始终通电，意味着在切换核心域断电时其状态得以保留。

**POWMAN**

表示电源管理器（POWMAN）部分或全部寄存器状态已复位。

**电源状态**

指示核心电压域的通电/断电状态发生变化。

**快速双击**

指示 CHIP\_RESET.DOUBLE\_TAP 位被复位。

**救援**

指示 CHIP\_RESET.RESCUE\_FLAG 位发生变化。

### 7.3.2. 芯片级复位目标

芯片级复位适用于以下主要组件：

- SW-DP 和 RP-AP 调试组件
- 电源管理器的临时寄存器和启动寄存器
- 包括常通电计时器在内的电源管理器
- 电源状态（恢复到状态 **P0.0**，所有域均通电，详见第6.2.2节“电源状态”）
- 系统复位（任何芯片级复位都会触发 PSM（上电状态机），该状态机负责系统复位的序列，详见第7.4节“系统复位（上电状态机）”）
- 看门狗（由任何芯片级复位触发，包括看门狗本身触发的复位）

芯片级复位同时复位以下两个 CHIP\_RESET 寄存器标志：

- **CHIP\_RESET.DOUBLE\_TAP**：引导只读存储器（bootrom）可利用该标志检测连接至 RUN 引脚的按钮的双击操作，并进入 USB 或 UART 引导加载程序。参见 BOOT\_FLAGS1.DOUBLE\_TAP OTP 标志。
- **CHIP\_RESET.RESCUE\_FLAG**：该标志指示引导只读存储器暂停启动流程。引导只读存储器清除此标志以示确认。您可利用此功能在几乎任何状态下执行全系统复位（特别是所有系统时钟停止的状态），并在处理器重新执行导致异常状态的代码之前捕获处理器。

**i 注意**

当 SW-DP 与 RP-AP 解除复位后，您可使用它们执行低级调试操作，如救援复位或通过 SWD 强制上电。但访问其它调试硬件，如 Mem-AP，需要系统时钟保持运行。

**i 注意**

这些标志位于 POWMAN 寄存器空间的 CHIP\_RESET 寄存器内，因此包含于常时供电（AON）电源域。

### 7.3.3. 芯片级复位源

按照严重程度，以下事件可能触发芯片级复位：

**上电复位（POR）**

上电复位确保芯片在首次通电时能干净启动，通过将芯片保持在复位状态，直到数字核心电源（DVDD）达到足以可靠驱动核心逻辑的电压水平。POR 组件详见第 7.6.1 节，“上电复位（POR）”。

## 欠压检测（BOD）

欠压检测器防止数字核心电源（DVDD）降至不安全工作水平以下而导致不可靠操作。BOD 组件详见第 7.6.2 节，“欠压检测（BOD）”。由 BOD 断言的复位称为欠压复位，简称 BOR。

### 外部复位

将 **RUN** 引脚拉低可复位芯片。此操作无论核心电源（DVDD）、上电复位模块或欠压检测模块状态如何，均能保持芯片处于复位状态。**RUN** 可用于延长初始上电复位时间，或由外部信号驱动，以根据需要启动和停止芯片。若未使用 **RUN** 信号，应将其拉高。双击将 **RUN** 设为低电平会设置 CHIP\_RESET.DOUBLE\_TAP 标志。引导代码读取该标志，若被设置则选择备用启动序列。

### 调试器复位请求

调试器可通过 **CDBGWRUPREQ** 控制发起芯片级复位。详情请参见第 3.5 节，“调试”。

### 救援调试端口复位

芯片也可通过救援调试端口复位，从而实现从死锁状态恢复。

救援调试端口复位不仅会复位芯片，还会设置 CHIP\_RESET.RESCUE\_FLAG。启动代码在启动时检查该标志，若被设置则进入安全状态。详见第 3.5.8 节，“救援复位”。

### 看门狗

看门狗通过在 WDSEL 寄存器中设置相应位，可触发不同级别的芯片级复位。由看门狗复位触发的芯片级复位将重置看门狗及看门狗擦写寄存器。POWMAN 中提供了额外的通用擦写寄存器。这些寄存器不会因看门狗触发的芯片级复位而被重置。

### SWCORE 电源关闭

有关关闭切换核心电源域（SWCORE）并触发此复位的操作列表，请参见  
第 6.2 节，“电源管理”。

### 瞬态故障检测器

若检测到 SWCORE 电源供应中的瞬态故障，将触发此复位。详情请参见第 10.9 节，“瞬态故障检测器”。

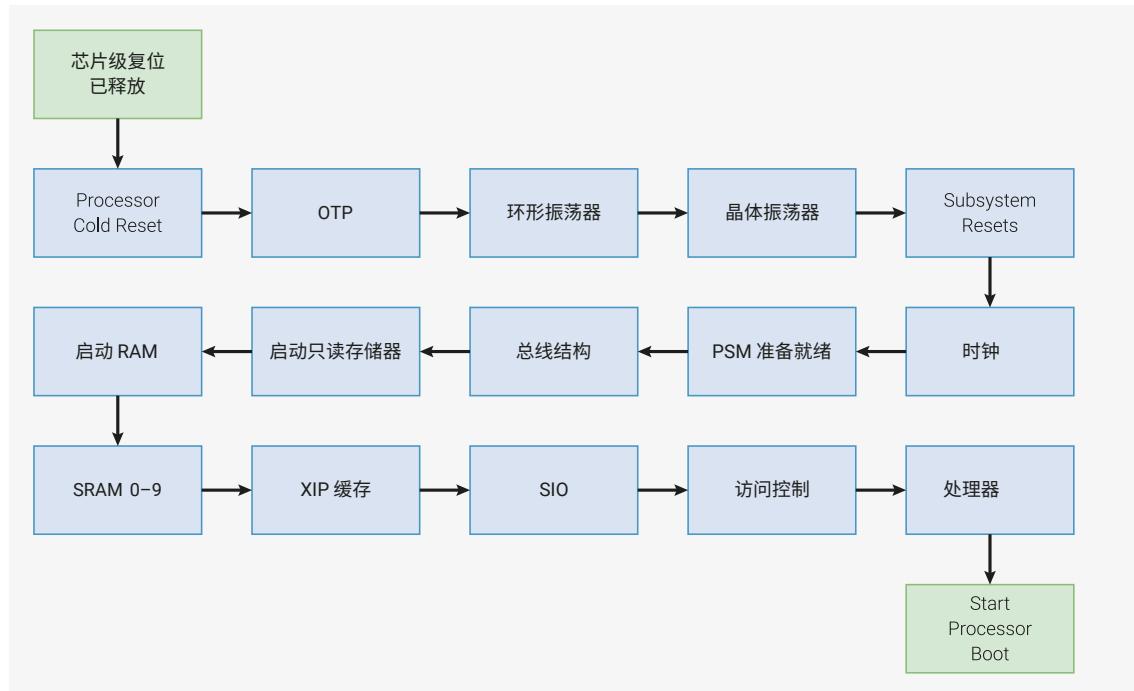
### RISC-V 非调试模块复位

RISC-V 调试模块中的 **dmcontrol.ndmreset** 位会复位系统中所有 RISC-V hart。该操作不会复位其他硬件。但它会作为芯片级复位原因记录于 CHIP\_RESET.HAD\_HZD\_SYS\_RESET\_REQ 中。有关 RISC-V 调试子系统的详细信息，请参见第 3.5.3 节，“RISC-V 调试”。

最后一次芯片级复位的来源记录于 CHIP\_RESET 寄存器。

完整的 POWMAN 寄存器列表详见第 6.4 节，“电源管理（POWMAN）寄存器”。

## 7.4. 系统复位（上电状态机）

图27。上电  
状态机  
序列

系统复位适用于处理器运行所必需的组件。系统组件之间存在相互依赖性，因此它们的重置由上电状态机（PSM）按顺序解除断言。每个序列器阶段完成时会输出复位完成信号 `rst_done`，并释放下一阶段的复位输入。部分序列将在写入 `FRCE_OFF` 寄存器或看门狗超时后执行。请注意，`FRCE_ON` 寄存器仅供内部使用，且在生产设备中已被禁用。

上电状态机在切换核心电源域上电后，按顺序释放系统级复位。其不同于控制电源域切换的电源管理器（POWMAN），详见第6.2节“电源管理”。

### 7.4.1. 复位序列

芯片级复位后，上电状态机（PSM）执行以下操作：

1. 解除处理器的冷复位。
2. 解除OTP复位。OTP读取启动所需的所有内容并断言 `rst_done`。
3. 启动环形振荡器。一旦振荡器输出稳定，断言 `rst_done`。
4. 解除晶体振荡器（XOSC）控制器复位。XOSC尚未启动，因此立即断言 `rst_done`。
5. 撤销主子系统复位，但不解除各子系统复位。
6. 启动 `clk_ref` 和 `clk_sys` 时钟生成器。在初始配置中，`clk_ref` 由无分频环形振荡器驱动，`clk_sys` 由 `clk_ref` 驱动。
7. PSM确认时钟处于激活状态。
8. 取消总线结构复位并初始化逻辑。
9. 取消各类内存控制器复位并初始化逻辑。
10. 取取消单周期IO子系统（SIO）复位并初始化逻辑。
11. 取消访问控制器复位并初始化逻辑。
12. 释放处理器复合体复位信号。核0和核1均从只读存储器（ROM）执行启动代码。启动代码读取核ID，核1进入休眠，核0继续执行引导程序。

看门狗复位触发后，PSM从PSM WDSEL寄存器指定的位置重新启动。

### 7.4.2. 寄存器控制

PSM是完全自动化的硬件组件：无需用户输入即可工作。调试器可通过写入 **FRCE\_ON** 寄存器触发完整或部分序列。**FRCE\_OFF** 寄存器为开发特性，生产设备中无效。

### 7.4.3. 与看门狗的交互

看门狗可通过写入 **WDSEL** 寄存器触发完整或部分序列。

### 7.4.4. 寄存器列表

PSM 寄存器起始地址为 **0x40018000**（在 SDK 中定义为 PSM\_BASE）。

表 529. PSM 寄存器列表

偏移量	名称	说明
0x0	<b>FRCE_ON</b>	强制解除复位（即开启电源）
0x4	<b>FRCE_OFF</b>	强制进入复位（即关闭电源）
0x8	<b>WDSEL</b>	如果看门狗需重置此项，则置为 1
0xc	<b>完成</b>	子系统是否已就绪？

### PSM: FRCE\_ON 寄存器

偏移：0x0

#### 描述

强制解除复位（即开启电源）

表 530. FRCE\_ON 寄存器

位	描述	类型	复位
31:25	保留。	-	-
24	<b>PROC1</b>	读写	0x0
23	<b>PROC0</b>	读写	0x0
22	<b>访问控制</b>	读写	0x0
21	<b>SIO</b>	读写	0x0
20	<b>XIP</b>	读写	0x0
19	<b>SRAM9</b>	读写	0x0
18	<b>SRAM8</b>	读写	0x0
17	<b>SRAM7</b>	读写	0x0
16	<b>SRAM6</b>	读写	0x0
15	<b>SRAM5</b>	读写	0x0
14	<b>SRAM4</b>	读写	0x0
13	<b>SRAM3</b>	读写	0x0
12	<b>SRAM2</b>	读写	0x0

位	描述	类型	复位
11	<b>SRAM1</b>	读写	0x0
10	<b>SRAM0</b>	读写	0x0
9	<b>BOOTRAM</b>	读写	0x0
8	<b>只读存储器</b>	读写	0x0
7	<b>总线架构</b>	读写	0x0
6	<b>PSM_READY</b>	读写	0x0
5	<b>时钟</b>	读写	0x0
4	<b>复位</b>	读写	0x0
3	<b>XOSC</b>	读写	0x0
2	<b>ROSC</b>	读写	0x0
1	<b>OTP</b>	读写	0x0
0	<b>PROC_COLD</b>	读写	0x0

## PSM: FRCE\_OFF 寄存器

偏移: 0x4

### 描述

强制进入复位（即关闭电源）

表 531. FRCE\_OFF  
寄存器

位	描述	类型	复位
31:25	保留。	-	-
24	<b>PROC1</b>	读写	0x0
23	<b>PROC0</b>	读写	0x0
22	<b>访问控制</b>	读写	0x0
21	<b>SIO</b>	读写	0x0
20	<b>XIP</b>	读写	0x0
19	<b>SRAM9</b>	读写	0x0
18	<b>SRAM8</b>	读写	0x0
17	<b>SRAM7</b>	读写	0x0
16	<b>SRAM6</b>	读写	0x0
15	<b>SRAM5</b>	读写	0x0
14	<b>SRAM4</b>	读写	0x0
13	<b>SRAM3</b>	读写	0x0
12	<b>SRAM2</b>	读写	0x0
11	<b>SRAM1</b>	读写	0x0
10	<b>SRAM0</b>	读写	0x0
9	<b>BOOTRAM</b>	读写	0x0
8	<b>只读存储器</b>	读写	0x0

位	描述	类型	复位
7	<b>总线架构</b>	读写	0x0
6	<b>PSM_READY</b>	读写	0x0
5	<b>时钟</b>	读写	0x0
4	<b>复位</b>	读写	0x0
3	<b>XOSC</b>	读写	0x0
2	<b>ROSC</b>	读写	0x0
1	<b>OTP</b>	读写	0x0
0	<b>PROC_COLD</b>	读写	0x0

## PSM: WDSEL 寄存器

偏移: 0x8

### 描述

如果看门狗需重置此项，则置为 1

表 532. WDSEL  
寄存器

位	描述	类型	复位
31:25	保留。	-	-
24	<b>PROC1</b>	读写	0x0
23	<b>PROC0</b>	读写	0x0
22	<b>访问控制</b>	读写	0x0
21	<b>SIO</b>	读写	0x0
20	<b>XIP</b>	读写	0x0
19	<b>SRAM9</b>	读写	0x0
18	<b>SRAM8</b>	读写	0x0
17	<b>SRAM7</b>	读写	0x0
16	<b>SRAM6</b>	读写	0x0
15	<b>SRAM5</b>	读写	0x0
14	<b>SRAM4</b>	读写	0x0
13	<b>SRAM3</b>	读写	0x0
12	<b>SRAM2</b>	读写	0x0
11	<b>SRAM1</b>	读写	0x0
10	<b>SRAM0</b>	读写	0x0
9	<b>BOOTRAM</b>	读写	0x0
8	<b>只读存储器</b>	读写	0x0
7	<b>总线架构</b>	读写	0x0
6	<b>PSM_READY</b>	读写	0x0
5	<b>时钟</b>	读写	0x0
4	<b>复位</b>	读写	0x0

位	描述	类型	复位
3	<b>XOSC</b>	读写	0x0
2	<b>ROSC</b>	读写	0x0
1	<b>OTP</b>	读写	0x0
0	<b>PROC_COLD</b>	读写	0x0

## PSM: DONE 寄存器

偏移量: 0xc

### 说明

子系统是否已就绪?

表 533. DONE  
寄存器

位	描述	类型	复位
31:25	保留。	-	-
24	<b>PROC1</b>	只读	0x0
23	<b>PROC0</b>	只读	0x0
22	<b>访问控制</b>	只读	0x0
21	<b>SIO</b>	只读	0x0
20	<b>XIP</b>	只读	0x0
19	<b>SRAM9</b>	只读	0x0
18	<b>SRAM8</b>	只读	0x0
17	<b>SRAM7</b>	只读	0x0
16	<b>SRAM6</b>	只读	0x0
15	<b>SRAM5</b>	只读	0x0
14	<b>SRAM4</b>	只读	0x0
13	<b>SRAM3</b>	只读	0x0
12	<b>SRAM2</b>	只读	0x0
11	<b>SRAM1</b>	只读	0x0
10	<b>SRAM0</b>	只读	0x0
9	<b>BOOTRAM</b>	只读	0x0
8	<b>只读存储器</b>	只读	0x0
7	<b>总线架构</b>	只读	0x0
6	<b>PSM_READY</b>	只读	0x0
5	<b>时钟</b>	只读	0x0
4	<b>复位</b>	只读	0x0
3	<b>XOSC</b>	只读	0x0
2	<b>ROSC</b>	只读	0x0
1	<b>OTP</b>	只读	0x0
0	<b>PROC_COLD</b>	只读	0x0

## 7.5. 子系统复位

### 7.5.1. 概述

复位控制器允许软件重置 RP2350 中的非关键组件。复位控制器可重置以下组件：

- USB 控制器
- 可编程输入输出端口
- 外围设备，包括 UART、I2C、SPI、PWM、定时器及 ADC
- PLLs
- IO和引脚寄存器

有关可通过复位控制器复位的所有组件完整列表，请参阅寄存器说明（第7.5.3节，“[寄存器列表](#)”）。

组件在复位时会于上电时保持复位状态。要使用该组件，软件必须撤销复位信号的断言。

#### 注意

SDK在复位后会自动撤销部分组件的复位断言。

### 7.5.2. 程序员模型

SDK使用以下结构体表示复位寄存器：

SDK链接：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware\\_structs/include/hardware/structs/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware_structs/include/hardware/structs/resets.h) 第63至159行

```

63 typedef struct {
64 _REG_(RESETS_RESET_OFFSET) // RESETS_RESET
65 // 0x10000000 [28] USBCTRL (1)
66 // 0x08000000 [27] UART1 (1)
67 // 0x04000000 [26] UART0 (1)
68 // 0x02000000 [25] TRNG (1)
69 // 0x01000000 [24] 定时器1 (1)
70 // 0x00800000 [23] 定时器0 (1)
71 // 0x00400000 [22] TBMAN (1)
72 // 0x00200000 [21] 系统信息 (1)
73 // 0x00100000 [20] 系统配置 (1)
74 // 0x00080000 [19] SPI1 (1)
75 // 0x00040000 [18] SPI0 (1)
76 // 0x00020000 [17] SHA256 (1)
77 // 0x00010000 [16] PWM (1)
78 // 0x00008000 [15] USB PLL (1)
79 // 0x00004000 [14] 系统 PLL (1)
80 // 0x00002000 [13] PIO2 (1)
81 // 0x00001000 [12] PIO1 (1)
82 // 0x00000800 [11] PIO0 (1)
83 // 0x00000400 [10] QSPI 引脚 (1)
84 // 0x00000200 [9] PADS_BANK0 (1)
85 // 0x00000100 [8] JTAG (1)
86 // 0x00000080 [7] IO_QSPI (1)
87 // 0x00000040 [6] IO_BANK0 (1)
88 // 0x00000020 [5] I2C1 (1)
89 // 0x00000010 [4] I2C0 (1)
90 // 0x00000008 [3] HSTX (1)

```

```

91 // 0x00000004 [2] DMA (1)
92 // 0x00000002 [1] BUSCTRL (1)
93 // 0x00000001 [0] ADC (1)
94 io_rw_32 reset;
95
96 _REG_(RESETS_WDSEL_OFFSET) // RESETS_WDSEL
97 // 0x1000000 [28] USBCTRL (0)
98 // 0x0800000 [27] UART1 (0)
99 // 0x0400000 [26] UART0 (0)
100 // 0x0200000 [25] TRNG (0)
101 // 0x0100000 [24] 定时器1 (0)
102 // 0x0080000 [23] 定时器0 (0)
103 // 0x0040000 [22] TBMAN (0)
104 // 0x0020000 [21] 系统信息 (0)
105 // 0x0010000 [20] 系统配置 (0)
106 // 0x0008000 [19] SPI1 (0)
107 // 0x0004000 [18] SPI0 (0)
108 // 0x0002000 [17] SHA256 (0)
109 // 0x0001000 [16] PWM (0)
110 // 0x00008000 [15] USB PLL (0)
111 // 0x00004000 [14] 系统PLL (0)
112 // 0x00002000 [13] PIO2 (0)
113 // 0x00001000 [12] PIO1 (0)
114 // 0x00000800 [11] PIO0 (0)
115 // 0x00000400 [10] PADS_QSPI (0)
116 // 0x00000200 [9] PADS_BANK0 (0)
117 // 0x00000100 [8] JTAG (0)
118 // 0x00000080 [7] IO_QSPI (0)
119 // 0x00000040 [6] IO_BANK0 (0)
120 // 0x00000020 [5] I2C1 (0)
121 // 0x00000010 [4] I2C0 (0)
122 // 0x00000008 [3] HSTX (0)
123 // 0x00000004 [2] DMA (0)
124 // 0x00000002 [1] BUSCTRL (0)
125 // 0x00000001 [0] ADC (0)
126 io_rw_32 wdsel;
127
128 _REG_(RESETS_RESET_DONE_OFFSET) // RESETS_RESET_DONE
129 // 0x1000000 [28] USBCTRL (0)
130 // 0x0800000 [27] UART1 (0)
131 // 0x0400000 [26] UART0 (0)
132 // 0x0200000 [25] TRNG (0)
133 // 0x0100000 [24] 定时器1 (0)
134 // 0x0080000 [23] 定时器0 (0)
135 // 0x0040000 [22] TBMAN (0)
136 // 0x0020000 [21] 系统信息 (0)
137 // 0x0010000 [20] 系统配置 (0)
138 // 0x0008000 [19] SPI1 (0)
139 // 0x0004000 [18] SPI0 (0)
140 // 0x0002000 [17] SHA256 (0)
141 // 0x0001000 [16] PWM (0)
142 // 0x00008000 [15] USB PLL (0)
143 // 0x00004000 [14] 系统PLL (0)
144 // 0x00002000 [13] PIO2 (0)
145 // 0x00001000 [12] PIO1 (0)
146 // 0x00000800 [11] PIO0 (0)
147 // 0x00000400 [10] PADS_QSPI (0)
148 // 0x00000200 [9] PADS_BANK0 (0)
149 // 0x00000100 [8] JTAG (0)
150 // 0x00000080 [7] IO_QSPI (0)
151 // 0x00000040 [6] IO_BANK0 (0)
152 // 0x00000020 [5] I2C1 (0)
153 // 0x00000010 [4] I2C0 (0)
154 // 0x00000008 [3] HSTX (0)

```

```

155 // 0x00000004 [2] DMA (0)
156 // 0x00000002 [1] BUSCTRL (0)
157 // 0x00000001 [0] ADC (0)
158 io_ro_32 reset_done;
159 } resets_hw_t;

```

该结构体定义以下寄存器：

- **reset**: 该寄存器包含每个可复位组件的对应位。设置为 1 时，复位信号被置位。若该位清零，复位信号撤销。
- **wdsel**: 该寄存器包含每个可复位组件的对应位。设置为 1 时，当看门狗触发，此组件将被复位。若复位上电状态机，则整个复位控制器（包括所有组件）将被复位。
- **reset\_done**: 该寄存器包含每个组件在退出复位后自动置位的状态位。软件可以通过等待该状态位来确保组件在使用前完成初始化。

SDK 定义了以下复位函数：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_resets/include/hardware/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h) 第159至161行

```

159 static __force_inline void reset_block(uint32_t bits) {
160 reset_block_mask(bits);
161 }

```

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_resets/include/hardware/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h) 第163至165行

```

163 static __force_inline void unreset_block(uint32_t bits) {
164 unreset_block_mask(bits);
165 }

```

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_resets/include/hardware/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h) 第167至169行

```

167 static __force_inline void unreset_block_wait(uint32_t bits) {
168 return unreset_block_mask_wait_blocking(bits);
169 }

```

重置函数的一个示例用法是UART驱动程序，该驱动程序定义了一个 **uart\_reset** 函数，根据指定的UART选择重置寄存器的不同位：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_uart/uart.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_uart/uart.c) 第32至38行

```

32 static inline void uart_reset(uart_inst_t *uart) {
33 reset_block_num(uart_get_reset_num(uart));
34 }
35
36 static inline void uart_unreset(uart_inst_t *uart) {
37 unreset_block_num_wait_blocking(uart_get_reset_num(uart));
38 }

```

### 7.5.3. 寄存器列表

重置控制器寄存器的基址为 **0x40020000**（在SDK中定义为RESETS\_BASE）。

表534.  
RESETS寄存器列表

偏移量	名称	说明
0x0	RESET	
0x4	WDSEL	
0x8	RESET_DONE	

## RESETS: RESET寄存器

偏移: 0x0

表535. RESET  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<b>USBCTRL</b>	读写	0x1
27	<b>UART1</b>	读写	0x1
26	<b>UART0</b>	读写	0x1
25	<b>TRNG</b>	读写	0x1
24	<b>TIMER1</b>	读写	0x1
23	<b>TIMER0</b>	读写	0x1
22	<b>TBMAN</b>	读写	0x1
21	<b>SYSINFO</b>	读写	0x1
20	<b>SYSCFG</b>	读写	0x1
19	<b>SPI1</b>	读写	0x1
18	<b>SPI0</b>	读写	0x1
17	<b>SHA256</b>	读写	0x1
16	<b>PWM</b>	读写	0x1
15	<b>PLL_USB</b>	读写	0x1
14	<b>PLL_SYS</b>	读写	0x1
13	<b>PIO2</b>	读写	0x1
12	<b>PIO1</b>	读写	0x1
11	<b>PIO0</b>	读写	0x1
10	<b>PADS_QSPI</b>	读写	0x1
9	<b>PADS_BANK0</b>	读写	0x1
8	<b>JTAG</b>	读写	0x1
7	<b>IO_QSPI</b>	读写	0x1
6	<b>IO_BANK0</b>	读写	0x1
5	<b>I2C1</b>	读写	0x1
4	<b>I2C0</b>	读写	0x1
3	<b>HSTX</b>	读写	0x1
2	<b>DMA</b>	读写	0x1
1	<b>BUSCTRL</b>	读写	0x1

位	描述	类型	复位
0	<b>ADC</b>	读写	0x1

## 复位：WDSEL 寄存器

偏移量：0x4

表 536. WDSEL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<b>USBCTRL</b>	读写	0x0
27	<b>UART1</b>	读写	0x0
26	<b>UART0</b>	读写	0x0
25	<b>TRNG</b>	读写	0x0
24	<b>TIMER1</b>	读写	0x0
23	<b>TIMERO</b>	读写	0x0
22	<b>TBMAN</b>	读写	0x0
21	<b>SYSINFO</b>	读写	0x0
20	<b>SYSCFG</b>	读写	0x0
19	<b>SPI1</b>	读写	0x0
18	<b>SPI0</b>	读写	0x0
17	<b>SHA256</b>	读写	0x0
16	<b>PWM</b>	读写	0x0
15	<b>PLL_USB</b>	读写	0x0
14	<b>PLL_SYS</b>	读写	0x0
13	<b>PIO2</b>	读写	0x0
12	<b>PIO1</b>	读写	0x0
11	<b>PIO0</b>	读写	0x0
10	<b>PADS_QSPI</b>	读写	0x0
9	<b>PADS_BANK0</b>	读写	0x0
8	<b>JTAG</b>	读写	0x0
7	<b>IO_QSPI</b>	读写	0x0
6	<b>IO_BANK0</b>	读写	0x0
5	<b>I2C1</b>	读写	0x0
4	<b>I2C0</b>	读写	0x0
3	<b>HSTX</b>	读写	0x0
2	<b>DMA</b>	读写	0x0
1	<b>BUSCTRL</b>	读写	0x0
0	<b>ADC</b>	读写	0x0

## 复位：RESET\_DONE 寄存器

偏移量: 0x8

表 537.  
RESET\_DONE 寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<b>USBCTRL</b>	只读	0x0
27	<b>UART1</b>	只读	0x0
26	<b>UART0</b>	只读	0x0
25	<b>TRNG</b>	只读	0x0
24	<b>TIMER1</b>	只读	0x0
23	<b>TIMERO</b>	只读	0x0
22	<b>TBMAN</b>	只读	0x0
21	<b>SYSINFO</b>	只读	0x0
20	<b>SYSCFG</b>	只读	0x0
19	<b>SPI1</b>	只读	0x0
18	<b>SPI0</b>	只读	0x0
17	<b>SHA256</b>	只读	0x0
16	<b>PWM</b>	只读	0x0
15	<b>PLL_USB</b>	只读	0x0
14	<b>PLL_SYS</b>	只读	0x0
13	<b>PIO2</b>	只读	0x0
12	<b>PIO1</b>	只读	0x0
11	<b>PIO0</b>	只读	0x0
10	<b>PADS_QSPI</b>	只读	0x0
9	<b>PADS_BANK0</b>	只读	0x0
8	<b>JTAG</b>	只读	0x0
7	<b>IO_QSPI</b>	只读	0x0
6	<b>IO_BANK0</b>	只读	0x0
5	<b>I2C1</b>	只读	0x0
4	<b>I2C0</b>	只读	0x0
3	<b>HSTX</b>	只读	0x0
2	<b>DMA</b>	只读	0x0
1	<b>BUSCTRL</b>	只读	0x0
0	<b>ADC</b>	只读	0x0

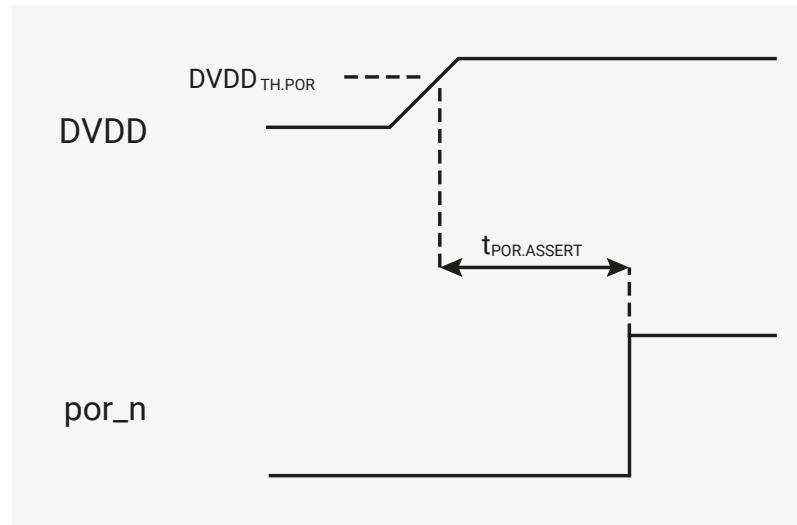
## 7.6. 上电复位与欠压检测

### 7.6.1. 上电复位 (POR)

上电复位模块确保芯片在首次加电时能够稳定启动。该模块通过保持芯片处于复位状态，直到数字核供电 ( $DVDD$ ) 达到足以可靠驱动芯片核心逻辑的电压水平来实现此目的。

该模块将其  $por\_n$  输出维持在低电平，直到  $DVDD$  超过上电复位阈值( $DVDD_{TH.POR}$ )，且持续时间超过上电复位断言延迟( $t_{POR ASSERT}$ )。一旦 $por\_n$ 变为高电平，即使 $DVDD$ 随后降至低于 $DVDD_{TH.POR}$ ，也会保持高电平。 $por\_n$ 在加电过程中的行为如图28所示，“一次上电复位周期”。

图28. 上电  
复位周期



$DVDD_{TH.POR}$ 固定为标称0.957V，阈值应介于0.924V与0.99V之间。该阈值假设  $DVDD$ 在初始上电时为标称1.1V，若使用较低电压， $por\_n$ 可能永远不会变为高电平。芯片脱离复位后， $DVDD$ 可以降低，而  $por\_n$ 不会变为低电平。

#### 7.6.1.1. 详细规格

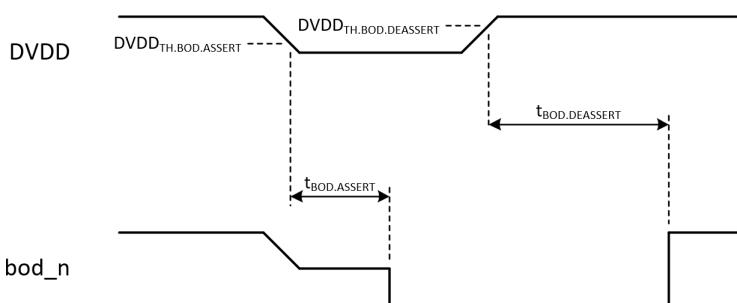
表538. 上电  
复位参数

参数	描述	最小值	典型值	最大值	单位
$DVDD_{TH.POR}$	上电复位 阈值	0.924	0.957	0.99	V
$t_{POR ASSERT}$	上电复位 断言延时		3	10	$\mu s$

### 7.6.2. 欠压检测 (BOD)

欠压检测模块防止数字核心电源 ( $DVDD$ ) 降至安全工作电压以下时发生不可靠操作。若启用，当  $DVDD$ 降至欠压检测断言阈值( $DVDD_{TH.BOD ASSERT}$ )以下，并持续时间超过欠压检测断言延时 ( $t_{BOD ASSERT}$ )时，模块通过将 $bod\_n$ 输出拉低来复位芯片。如果  $DVDD$ 随后上升超过棕褐色检测取消断言阈值 ( $DVDD_{TH.BOD DEASSERT}$ )且持续时间超过棕褐色检测取消断言延迟 ( $t_{BOD DEASSERT}$ )，该模块通过将 $bod\_n$ 拉高来释放复位。图29，“棕褐色检测周期”展示了一次棕褐色及其供电恢复过程。

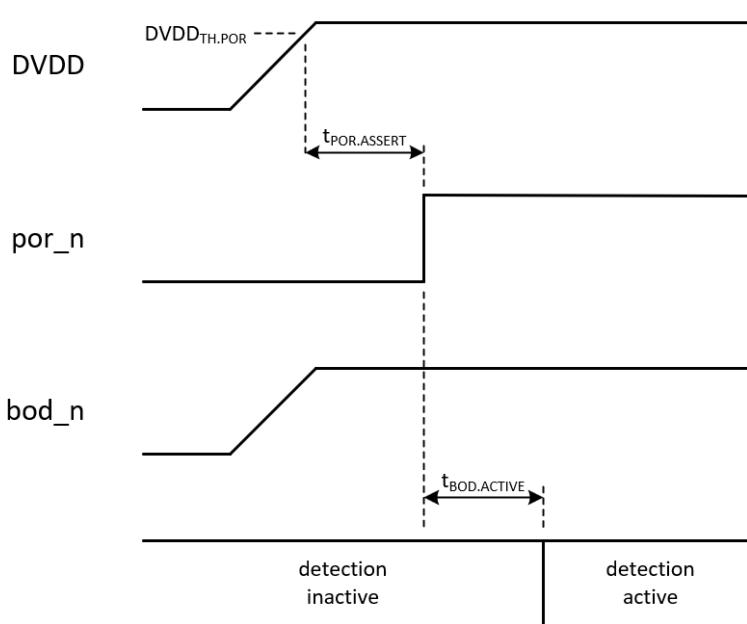
图29。棕褐色检测周期



### 7.6.2.1. 检测启用

棕褐色检测在设备初次上电时始终启用。但在por\_n拉高与检测激活之间存在一个短暂延迟，即棕褐色检测激活延迟 (tBOD.ACTIVE)。如图30，“设备初次上电及棕褐色事件后棕褐色检测的激活”所示。

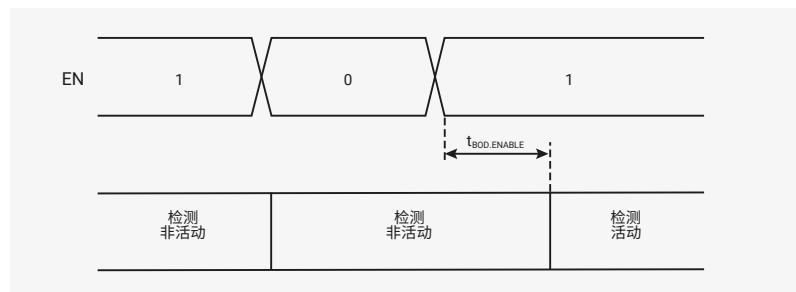
图30。设备初次上电及棕褐色事件后棕褐色检测的激活



芯片一旦解除复位，检测即可通过软件控制被禁用。此举可节省少量功耗。如果随后重新启用检测，将有一次短暂延迟，即棕色断电检测启用延迟 (t\_BOD\_ENABLE)，之后检测方会重新生效。该过程如图31所示，“禁用与启用棕色断电检测”。

通过向BOD寄存器的 EN字段写入0可禁用检测，写入 1则可重新启用检测。检测被禁用时，模块的 bod\_n输出为高电平。

图31。禁用与启用棕色断电检测



若BOD寄存器被复位，检测将重新启用，因为寄存器的 EN字段会被设置为 1。检测将

在等于棕色断电检测启用延迟 ( $t_{BOD.ENABLE}$ ) 的时间后重新生效。

#### **i 注意**

若 BOD 寄存器因上电或低电压复位触发而重置，寄存器重置与棕色断电检测激活之间的延迟将等于棕色断电检测激活延迟 ( $t_{BOD.ACTIVE}$ )。对于所有其他复位源，该延迟将等于棕色断电检测使能延迟 ( $t_{BOD.ENABLE}$ )。

### 7.6.2.2. 调整检测阈值

棕色断电检测阈值(DVDDTH.BOD) 在初次上电或复位事件后具有标称值0.946V。

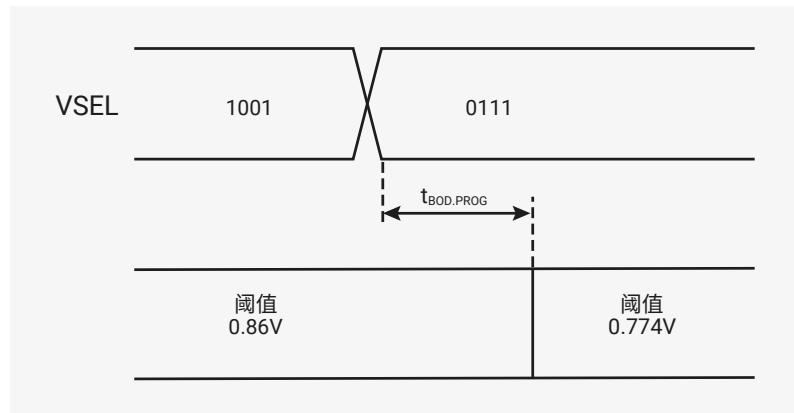
该值对应的检测阈值应在0.913V至0.979V之间。复位结束后，阈值可通过软件进行调整。新的检测阈值将在**棕色断电检测编程延迟** ( $t_{BOD.PROG}$ ) 后生效。示例见图32，“调整棕色断电检测阈值”。

通过向 BOD 寄存器中的 VSEL 字段写入数据调整阈值。详见 BOD 寄存器说明。

#### **i 注意**

DVDD的额定供电电压为1.1 V。请勿将欠压检测阈值设置高于额定供电电压。

图32。调整  
欠压检测  
阈值



### 7.6.2.3. 详细规格

表539。欠压检测参数

参数	描述	最小值	典型值	最大值	单位
$DVDD_{TH.BOD ASSERT}$	欠压检测断言阈值	96.5	100	103.5	所选阈值电压的百分比
$DVDD_{TH.BOD DEASSERT}$	欠压检测取消断言阈值	97.4	101	105	所选阈值电压的百分比
$t_{BOD.ACTIVE}$	欠电压检测激活延迟		55	80	μs
$t_{BOD ASSERT}$	欠电压检测断言延迟		3	10	μs

参数	描述	最小值	典型值	最大值	单位
$t_{BOD.DEASSERT}$	欠电压 检测解除断言 延迟		55	80	μs
$t_{BOD.ENABLE}$	欠电压 检测启用 延迟		35	55	μs
$t_{BOD.PROG}$	欠电压 检测 编程 延迟		20	30	μs

### 7.6.3. 电源监控器

上电和欠电压复位模块由核心电压调节器的模拟电源（VREG\_AVDD）供电。模块在首次加电时完成初始化，但若 VREG\_AVDD 未降至足够低的水平即断电再重新加电，可能无法可靠地重新初始化。为防止此类情况发生，监测 VREG\_AVDD，并在其低于**VREG\_AVDD激活阈值**（VREG\_AVDD<sub>TH\_ACTIVE</sub>）时重新初始化上电复位模块。

VREG\_AVDD<sub>TH\_ACTIVE</sub>固定为标称1.1V，阈值范围应为0.87V至1.26V。该阈值不表示安全工作电压。此电压表示 VREG\_AVDD 必须下降到的阈值以下，以可靠地重新初始化上电复位模块。为确保安全运行，**VREG\_AVDD的标称电压应为3.3V**。详见表144 1，“电源规格”。

#### 7.6.3.1. 详细规格

表540。电压  
稳压器输入供电  
监测参数

参数	描述	最小值	典型值	最大值	单位
VREG_VIN <sub>TH_ACTIVE</sub>	<b>VREG_VIN</b> 激活阈值	0.87	1.1	1.26	V

### 7.6.4. 寄存器列表

芯片级复位子系统与常通电域内其他电源管理子系统共享寄存器地址空间。该地址空间在本文档中称为 **POWMAN**。**POWMAN** 寄存器完整列表见第6.4节“电源管理（POWMAN）寄存器”，但与棕断检测器相关的寄存器信息在此重复列出。

**POWMAN**寄存器基址为0x40100000（在SDK中定义为POWMAN\_BASE）。

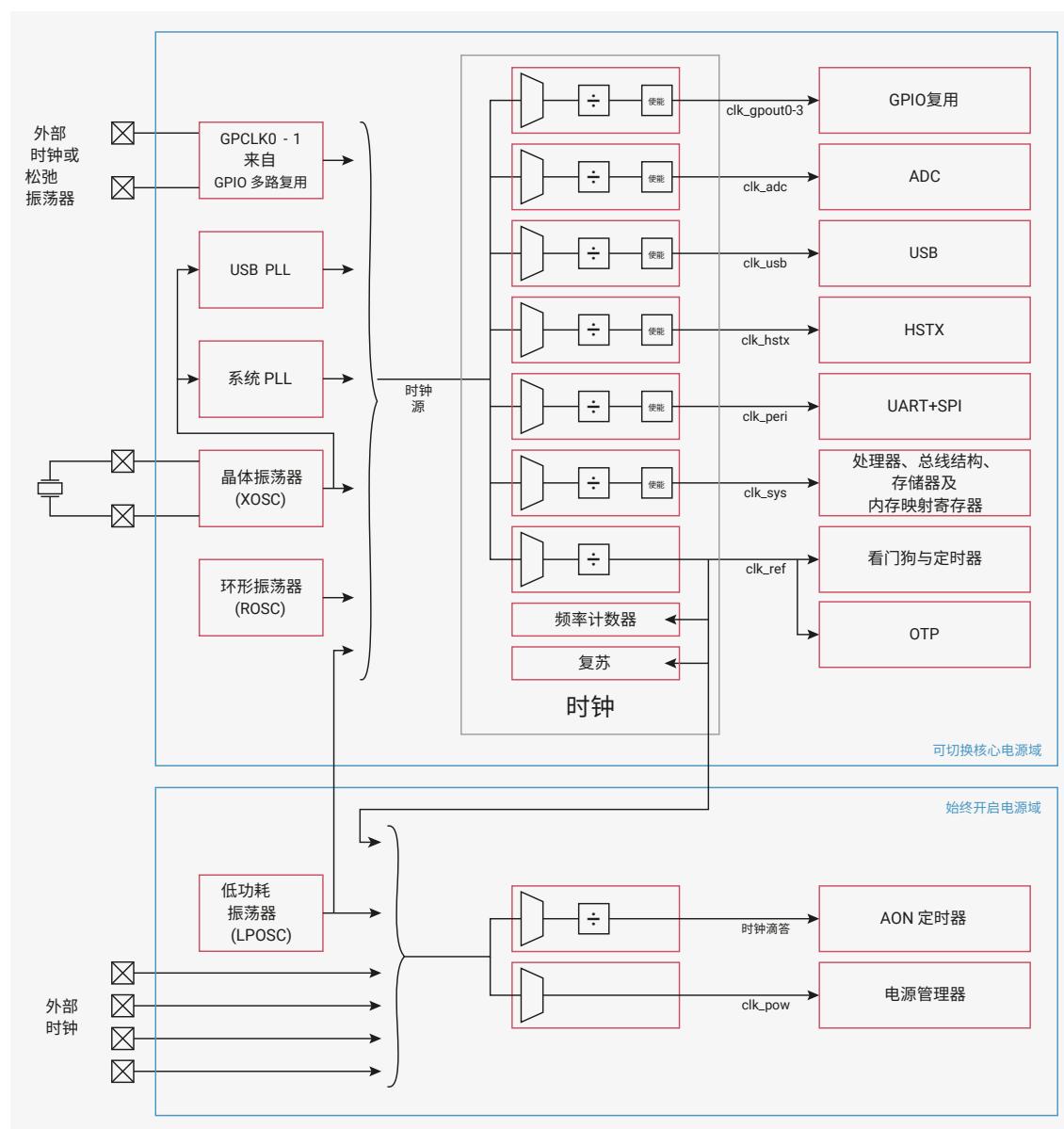
- [BOD\\_CTRL](#)
- [BOD](#)
- [BOD\\_LP\\_ENTRY](#)
- [BOD\\_LP\\_EXIT](#)

# 第8章 时钟

## 8.1. 概述

时钟模块为片上及外部组件提供独立时钟。其接收多种时钟源输入，使用户能够在性能、成本、板载面积及功耗之间进行权衡。该模块利用多个时钟发生器，从这些时钟源中产生所需时钟。该架构允许用户灵活地独立启动和停止时钟，并在保持部分时钟于最优频率的同时调整其他时钟频率。

图33. 时钟概览



晶体振荡器 (XOSC) 为两个锁相环 (PLL) 提供参考，这些PLL为处理器和外设提供高精度时钟。在从各种低功耗模式唤醒时，这些时钟启动较慢，因此片上环形振荡器 (ROSC) 用于启动设备，直到它们可用为止。当切换核心断电或设备处于DORMANT模式（参见第6.5.3节“DORMANT状态”）时，片上32kHz低功耗振荡器 (LPOSC) 为电源管理器提供时钟，并为始终开启定时器 (AON定时器) 提供滴答信号。

时钟发生器从时钟源中选择，并可选择性地对选中的时钟进行分频，随后通过使能逻辑输出，该逻辑在睡眠模式下实现自动时钟门控（参见第8.1.3.5.2节“系统睡眠模式”）。

片上频率计有助于调试时钟设置，同时允许测量LPOSC、ROSC及外部时钟的频率。若系统时钟意外停止，片上重启(resume，意为复苏)模块将从已知的正常时钟重新启动系统时钟。此功能使软件调试器能够访问寄存器并排查问题。

切换核供电时，电源管理时钟会自动切换至参考时钟（`clk_ref`）。

用户可选择切换AON定时器计时，但建议等待 `clk_ref`由XOSC驱动稳定后再切换，因ROSC频率存在不精确性。

您可使用最多两个GPIO时钟输入替代默认时钟源。此举有助于避免在已有精确时钟源的系统中增加第二晶体，并能用更准确的外部时钟替代ROSC和LPOSC。

您还可以将最多4个生成时钟以最高50MHz频率输出至GPIO。此功能使您能够向外部设备提供时钟信号，减少对额外时钟组件的需求，从而降低功耗和电路板面积。

### 8.1.1. RP2350 版本间的变更

RP2350 A3更改了以下复位值：

- `CLK_SYS_CTRL.SRC` 从 0 变更为 1（选择AUX源）。
- `CLK_SYS_CTRL.AUXSRC` 从 0 变更为 2（选择ROSC作为AUX源）。

有关复位时对ROSC配置所做相关更改的信息，请参见硬件更改一节。有关A3版Boot ROM中相关更改的信息，请参见Boot ROM更改一节。

### 8.1.2. 时钟源

RP2350支持多种时钟源。该灵活性允许用户针对性能、成本、电路板面积及功耗优化时钟配置。RP2350支持以下潜在的时钟源：

- 片上32kHz低功耗振荡器（第8.4节，“低功耗振荡器（LPOSC）”）
- 片上环形振荡器（第8.3节，“环形振荡器（ROSC）”）
- 晶体振荡器（第8.2节，“晶体振荡器（XOSC）”）
- 来自GPIO（第8.1.6.4节，“配置GPIO输入时钟”）和PLL（第8.6节，“PLL”）的外部时钟

时钟源列表因时钟发生器不同而异，可在 `CTRL` 寄存器中以枚举值形式查阅。

以`CLK_SYS_CTRL`为例。

#### 8.1.2.1. 低功耗振荡器

片上32kHz低功耗振荡器（第8.4节，“低功耗振荡器（LPOSC）”）无需任何外部元件。

当始终通电域通电时，振荡器自动启动，为电源管理器提供时钟信号，并在切换核心电源域关闭时为始终通电定时器（AON定时器）提供计时脉冲。

该LPOSC频率可调至1%精度，且AON定时器计时脉冲发生器中的分频器可对1ms计时脉冲进行进一步微调。但由于LPOSC频率随电压和温度变化，故细调仅适用于电压和温度相对稳定的系统。

当切换核心通电时，LPOSC时钟可驱动参考时钟（`clk_ref`），进而驱动系统时钟（`clk_sys`）。这允许另一种低功耗模式，在该模式下处理器保持通电，但与SLEEP和DORMANT模式不同，时钟继续运行。LPOSC时钟亦可送至频率计进行校准，或输出至GPIO。

### 8.1.2.2. 环形振荡器

片上环形振荡器（第 8.3 节，“[环形振荡器（ROSC）](#)”）无需外部元件。当切换核心域通电时，环形振荡器自动启动，并用于芯片初始启动阶段的时钟驱动。

启动期间，ROSC 以标称 11MHz 频率运行，但会随 PVT（工艺、电压和温度）变化。ROSC 频率保证处于 4.6MHz 至 19.6MHz 范围内。

对于频率精度不重要的低成本应用，芯片可继续由 ROSC 运行。若您的应用需更高性能，可通过对寄存器编程提升频率，详见第 8.3 节“[环形振荡器（ROSC）](#)”。因频率随 PVT（工艺、电压及温度）变化，用户务必避免超过时钟发生器章节中规定的最大频率。关于在接近最大频率运行 ROSC 时减少频率波动的措施，请参阅第 8.1.2.2.1 节“[缓解工艺引起的 ROSC 频率变化](#)”。或者，可采用外部时钟或 XOSC 提供稳定参考时钟，并利用 PLL 生成更高频率。但此方案需要外部组件，会增加电路板面积并提高功耗。

使用外部时钟或 XOSC 时，可停止 ROSC 以节省功耗。停止 ROSC 前，必须将参考时钟发生器及系统时钟发生器切换至备用时钟源。

当切换核心域断电时，ROSC 处于无电状态，但切换核心上电时会立即启动。其运行不受睡眠模式影响。为节约功耗，请在进入睡眠模式前降低频率。

进入 DORMANT 模式时，ROSC 会自动停止。退出 DORMANT 模式时，ROSC 将以相同配置重新启动。若驱动 ROSC 时钟频率接近最大值，须在进入 SLEEP 或 DORMANT 模式前降低频率。此举可补偿 SLEEP 或 DORMANT 模式期间因环境条件变化引起的频率波动。

欲外部使用 ROSC 时钟，请通过 [clk\\_gpcclk0-3](#) 生成器将其输出至 GPIO 引脚。

下列章节阐述缓解 ROSC 频率 PVT 变化的技术方案。同时，这些内容为教学 PVT 效应及编写实时控制软件提供了重要的设计挑战。

#### 提示

由于 ROSC 频率随 PVT（工艺、电压和温度）变化，只要已知其中任意两个变量，便可利用 ROSC 频率测量第三个变量。

#### 8.1.2.2.1. 缓解因工艺造成的 ROSC 频率变化

工艺变化的原因如下：

- 芯片出厂时存在一定范围的工艺参数差异。这导致芯片间 ROSC 频率存在差异。
- 芯片随着老化，工艺参数会发生轻微变化。这一现象仅在运行数千小时后才可观测到。

为缓解工艺变异，用户可对单个芯片进行特性描述并相应编程 ROSC 频率。

此方法适用于少量芯片，但不适合大规模生产。对于大批量应用，应考虑采用自动缓解机制。

#### 8.1.2.2.2. 缓解因电压引起的 ROSC 频率变化

电源电压变化的原因如下：

- 电源本身可能存在波动。
- 由于芯片活动的变化，片上 IR 也随之变化。

为减轻电压变化，应校准应用的最低性能目标，然后调整 ROSC

频率，以始终超过该最低值。

### 8.1.2.2.3. 缓解因温度引起的ROSC频率变化

温度变化的原因包括：

- 环境温度可能发生变化。
- 由于芯片活动引起的自我发热，芯片温度也会变化。

为减轻温度变化，应保持温度稳定。您可以采用温控环境、被动散热或主动散热。或者，利用片上温度传感器监测温度，并调整ROSC频率，使其保持在所需范围内。

### 8.1.2.2.4. 基于PVT的ROSC频率变化自动缓解

自动ROSC频率控制技术避免了单个芯片校准的需要，但需定期访问时钟参考或时间参考。

如果有时钟参考，可利用其周期性测量ROSC频率并进行相应调整。片上XOSC是一种潜在的时钟参考源。对于极低功耗应用，您甚至可以间歇性启动XOSC以节省功耗，因为持续运行XOSC或使用PLL来实现高频率的代价过高。

如果具备时间参考，您可以使用ROSC为片上AON定时器提供时钟，并定期将其与时间参考进行比较，必要时调整ROSC频率。采用这些技术后，ROSC频率仍会因电压和温度变化而发生漂移。因此，您还应针对电压和温度变化实施缓解措施，以确保ROSC频率的漂移不会超出可接受范围。

### 8.1.2.2.5. 利用ROSC的自动超频

任何数字设备的数据手册中标注的最大频率均为最恶劣PVT条件下的数值。大多数芯片在大多数正常环境中能够显著超过该最大频率运行，从而支持超频。当RP2350工作于ROSC时，PVT因素同时影响ROSC与数字组件。随着ROSC频率的提升，处理器亦可运行得更快。这意味着用户可以通过ROSC进行超频，然后依赖ROSC频率随着PVT变化进行追踪。ROSC频率与处理器性能的追踪并非完美，且目前缺乏足够数据以确定该工作模式下的安全ROSC设置，因此需要进行适当实验。

该工作模式最大化处理器性能，但会导致完成任务所需时间出现波动。

仅在该时间波动可接受的应用场景中使用超频。如您的应用涉及频率敏感接口（如USB或UART），必须采用XOSC和PLL为相关组件提供精确时钟。

### 8.1.2.3. 晶体振荡器

晶体振荡器（见第8.2节“晶体振荡器（XOSC）”）提供精确且稳定的时钟参考，当需要精准定时且无合适外部时钟时应予以采用。XOSC需要外部晶体元件。外部晶体决定振荡频率。RP2350 支持 1MHz 至 50MHz 的晶体，且 RP2350 参考设计（参见 RP2350 硬件设计与最小设计示例）采用 12MHz 晶体。利用 XOSC 与 PLL，可使片上组件达到其最大工作频率。设计中预留了适当裕度，以容忍 XOSC 频率最高达 1000ppm 的偏差。

当切换核心域断电时，XOSC 不供电。切换核心上电时，XOSC 仍保持非活动状态。如需启用，必须通过软件激活。XOSC 启动需数毫秒，软件必须等待 `XOSC_STABLE` 标志置位后，方可启动 PLL 并调整任何时钟发生器。在 XOSC 启动完成前，输出信号可能不存在或仅呈现极短脉冲；若此时使用输出信号，将导致逻辑损坏。当 XOSC 启动完成后，参考时钟 (`clk_ref`) 和系统时钟 (`clk_sys`) 可由 XOSC 驱动运行。如果您

将系统和参考时钟切换为由XOSC驱动运行，可以停止ROSC以节省功耗。

XOSC不受睡眠模式影响。进入和退出休眠模式时，它会自动在相同配置下停止并重新启动。

若要将XOSC时钟输出到外部，需使用 `clk_gpclk0`至`clk_gpclk03`生成器之一，将其输出至GPIO引脚。不可直接从XIN（XI）或XOUT（XO）引脚获取XOSC输出。

#### 8.1.2.4. 外部时钟

如果硬件设计中存在外部时钟，可用于为RP2350提供时钟。时钟可单独使用，也可与其他（内部或外部）时钟源联合使用。使用XIN及GPIN0至GPIN1之一输入外部时钟。

如果您向XIN输入外部时钟，则无需外部晶体。在向XIN输入外部时钟时，必须配置XOSC以透传XIN信号。当切换核心断电时，该配置将会丢失，但该配置不受SLEEP和DORMANT模式的影响。输入信号限制为50MHz，但片内PLL可根据需要从XIN输入合成更高频率。

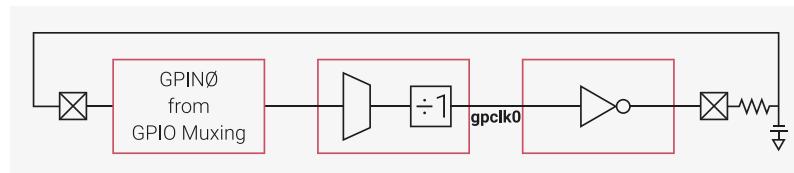
GPIN0-GPIN1可提供系统和外设时钟，但频率限于50MHz。这有助于节省功耗，并允许RP2350上的组件与外部组件同步运行，从而简化芯片间的数据传输。若外部时钟的频率精度低于1000ppm，生成的时钟不应在最大频率下运行，否则可能超出其设计裕度。一旦外部时钟开始运行，参考时钟（`clk_ref`）和系统时钟（`clk_sys`）即可由外部时钟驱动，您可以关闭ROSC以节省功耗。当切换核心断电时，GPIN0-GPIN1的配置将丢失，但该配置不受SLEEP与DORMANT模式的影响。

为了为AON计时器提供更精确的时钟，可以使用GPIN0-GPIN3中的任一输入替代LPOSC时钟。此类输入频率限制为29MHz。GPIN0-GPIN3的配置不受切换核心断电、睡眠模式及DORMANT模式的影响。

#### 8.1.2.5. 松弛振荡器

若无合适的时钟源，但仍需用其他时钟替代或补充外部时钟，则可使用外部无源元件构建一至两个松弛振荡器。将时钟源（GPIN0-GPIN1）发送至 `clk_gpclk0`至`clk_gpclk03`生成器之一，通过GPIO反相器`OUTOVER`反向，并通过RC电路连接回时钟源输入：

图34。简单  
松弛振荡器  
示例



由松弛振荡器产生的时钟频率取决于芯片的延迟和GPIO输出的驱动电流，这两者均受工艺、压、温（PVT）变化的影响。频率和频率准确度则依赖于外部元件的质量和精度。更复杂的外部元件，如陶瓷谐振器，可提升性能，但同时增加成本和系统复杂性。此类振荡器无法达到1000ppm的精度，因此无法以最大频率驱动内部时钟。若需以最高可能频率驱动内部时钟，请使用XOSC。

松弛振荡器的配置在切换核心断电时将丢失，但不受睡眠模式或休眠模式（DORMANT模式）影响。

### 8.1.2.6. PLL

锁相环（PLL，章节8.6，“PLL”）用于在使用XOSC或外部时钟源驱动至XIN引脚时提供高速时钟。在完整功能的应用中，USB PLL为ADC和USB提供固定的48MHz时钟，同时 `clk_ref`由XOSC或外部时钟源驱动。这允许用户通过系统PLL驱动`clk_sys`，并可根据需求调整频率以节省功耗，无需更改其他时钟的设置。

`clk_peri`可由固定频率的USB PLL或可变频率的系统PLL驱动。如`clk_sys`无需超过48MHz，则可仅使用一套PLL，且`clk_sys`时钟生成器中的分频器可根据需求调整`clk_sys`的频率。

PLL启动后，必须等待锁定，锁定状态由 `STATUS`寄存器中的`LOCK`位指示，在此之前不得使用PLL输出。因此，在更改参考时钟分频器、输出分频器或旁路模式期间，PLL输出不可用。反馈分频器更改时，输出可以继续使用，但在大幅调整反馈分频器时，输出频率可能会出现超调或欠调现象。详见第8.6节，“PLL”，以获取更多信息。

只要参考时钟精度达到1000ppm，PLL即可驱动时钟达到最大频率，确保生成时钟频率保持在设计容差范围内。

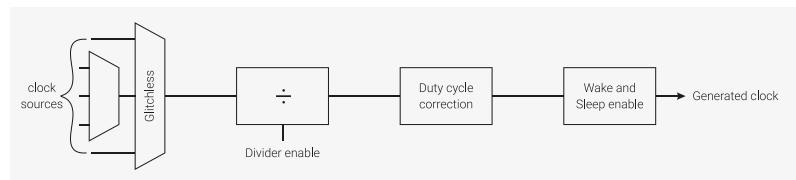
PLL不受睡眠模式影响。为节省睡眠模式下功耗，请将所有时钟发生器切换远离PLL，并在进入睡眠模式前通过软件停止PLL。

PLL在进入及退出休眠模式时不会自动停止或重启。若PLL在进入休眠模式时仍运行，因XOSC中的参考时钟停止，PLL将受到损坏。这会产生失控时钟，造成不必要的功耗。进入休眠模式前，须将所有时钟发生器切换远离PLL，并通过软件停止PLL。

### 8.1.3. 时钟发生器

时钟发生器采用标准设计，包含时钟源多路复用、分频、占空比校正及睡眠模式启用功能。为节省芯片面积与功耗，部分时钟发生器省略某些功能。

图35. 通用时钟发生器



#### 8.1.3.1. 实例

RP2350配置了多个时钟发生器，详列如下。

表541. RP2350  
时钟发生器

时钟	描述	标称频率
<code>clk_gpout0</code>	时钟输出至GPIO。可用于为外部设备提供时钟，或使用逻辑分析仪或示波器对芯片时钟进行调试。	不适用
<code>clk_gpout1</code>		
<code>clk_gpout2</code>		
<code>clk_gpout3</code>		
<code>clk_ref</code>	参考时钟，除非处于休眠模式（DORMANT），否则持续运行。上电时由环形振荡器（ROSC）驱动，但可切换至晶体振荡器（XOSC）以获得更高精度。	6 - 12MHz

时钟	描述	标称频率
<code>clk_sys</code>	系统时钟，除非处于休眠模式（DORMANT），否则持续运行。上电时由 <code>clk_ref</code> 驱动，通常切换为PLL。	150MHz
<code>clk_peri</code>	外设时钟。通常由 <code>clk_sys</code> 驱动，但当 <code>clk_sys</code> 经软件更改时，允许外设保持恒定速度运行。	12 - 150MHz
<code>clk_usb</code>	USB参考时钟，必须为48MHz。	48MHz
<code>clk_adc</code>	ADC参考时钟，必须为48MHz。	48MHz
<code>clk_hstx</code>	HSTX时钟。	150MHz

有关各时钟发生器所有时钟源的完整列表，请参阅相应的CTRL寄存器。例如，`CLK_SYS_CTRL`。

### 8.1.3.2. 复用器

所有时钟发生器均配备一个称为辅助（aux）复用器的复用器。该复用器采用传统设计，切换选择控制时输出会产生毛刺。参考时钟（`clk_ref`）和系统时钟（`clk_sys`）具有额外的复用器，称为无毛刺复用器（glitchless mux）。无毛刺复用器可在不产生输出毛刺的情况下切换时钟源。

切换辅助复用器的时钟源之前，必须执行以下任一操作：

- 临时将无毛刺复用器切换离开aux（若无毛刺复用器可用）。
- 临时通过其`CTRL_ENABLE`位禁用时钟发生器。
- 保持目的模块处于复位状态，以防止潜在时钟毛刺导致未定义操作。

未执行上述任一操作，可能导致该时钟发生器当前所有时钟输入出现毛刺。务必避免时钟毛刺；它们可能会破坏时钟运行的逻辑。

时钟发生器需要两个源时钟周期来停止输出，并需要两个新源时钟周期来重新启动输出。在更改辅助复用器之前，请等待发生器停止。当目标时钟比系统时钟慢得多时，存在软件在时钟发生器安全停止前修改辅助复用器源的风险。通过轮询时钟发生器的`CTRL_ENABLED`状态，直至其与`CTRL_ENABLE`的值匹配，以避免此类情况。

无毛刺复用器仅适用于始终开启的时钟。在RP2350上，始终开启的时钟是参考时钟（`clk_ref`）和系统时钟（`clk_sys`）。此类时钟必须持续运行，除非芯片处于休眠（DORMANT）模式。无毛刺复用器具有状态输出（`SELECTED`），用于指示所选时钟源。您可通过软件读取该状态输出，以确认时钟源切换已完成。

推荐的控制序列如下：

切换无毛刺多路复用器的时钟源：

- 将无毛刺多路复用器切换至备用源。
- 轮询 `SELECTED` 寄存器，直至切换完成。

当发生器配备无毛刺多路复用器时，切换辅助多路复用器的时钟源：

- 将无毛刺多路复用器切换至非辅助多路复用器的时钟源。
- 轮询 `SELECTED` 寄存器，直至切换完成。
- 更改辅助多路复用器的选择控制。

4. 将无毛刺多路复用器切换回辅助多路复用器。

5. 如有必要，轮询 **SELECTED** 寄存器，直至切换完成。

当发生器未配备无毛刺多路复用器时，切换辅助多路复用器的时钟源：

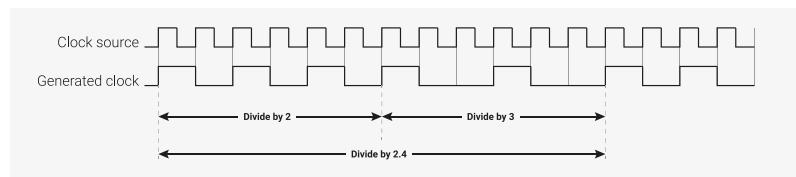
1. 禁用时钟分频器。
2. 等待生成的时钟停止（时钟源的两个周期）。
3. 更改辅助多路复用器的选择控制。
4. 启用时钟分频器。
5. 如有必要，等待时钟发生器重启（时钟源的两个周期）。

代码示例请参见第8.1.6.1节，“配置时钟发生器”。

### 8.1.3.3 分频器

功能齐全的分频器可对范围在1.0至 $2^{16}$ 之间的分数进行除法运算。分数除法通过在两个整数除数之间切换实现；这将产生抖动的时钟信号，可能不适用于某些应用。例如，当除以2.4时，分频器对三个周期使用除数2，对两个周期使用除数3。对于整数部分较大的除数，抖动将显著减小且影响较小。

图36. 分数除法的一个示例。



所有分频器均支持**动态除数切换**：输出时钟可实现从一个除数向另一个除数的平滑切换。由于分频器将除数更改同步至时钟周期结束，时钟发生器在更改除数时无需停止。同样，分频器将使能信号同步至时钟周期结束，避免时钟发生器使能或禁用时产生毛刺。始终开启的时钟发生器为永久激活状态，因此不设有使能控制。

若时钟发生器发生锁死且无法完成当前时钟周期，可通过 **KILL** 控制强制停止。此操作可能导致输出毛刺，从而破坏时钟驱动的逻辑。在使用 **KILL** 控制前，务必先重置目标逻辑。始终开启的时钟发生器为永久激活状态，因此不设有 **KILL** 控制。

#### ① 注意

该时钟发生器设计已被广泛应用于多款芯片，且从未出现锁死现象。因 **KILL** 控制设计粗糙且不必要，不应将其作为使能控制的替代方案。

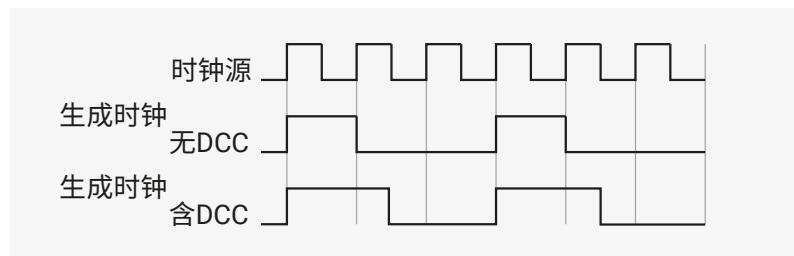
### 8.1.3.4. 占空比校正

分频器工作于输入时钟的上升沿，因此在除以奇数时无法产生均匀占空比的时钟信号。例如，除以3时，占空比为33.3%；除以5时，占空比为40%。

如果启用，占空比校正逻辑会将输出时钟的下降沿移至输入时钟的下降沿，并恢复50%的占空比。占空比校正可在时钟运行时启用或禁用。当进行偶数分频时，该功能不工作。

图37。占空比较正

示例



### 8.1.3.5. 时钟使能

每个时钟信号都分发至多个目的地。除少数例外，每个目的地设有两个使能信号。使用 `WAKE_EN` 寄存器在系统唤醒时启用时钟。使用 `SLEEP_EN` 寄存器在系统睡眠模式时启用时钟。使能信号有助于降低未使用组件时钟分配网络的功耗。任何未接收时钟的组件将保留其配置，以便快速重启。

#### ① 注意

默认情况下，`WAKE_EN` 和 `SLEEP_EN` 寄存器复位为 `0x1`，表示启用所有时钟。仅将此功能用于低功耗设计。

#### 8.1.3.5.1. 时钟使能例外

以下目标无时钟使能：

- 生成器 `clk_gpclk0-clk_gpclk03`。
- 处理器核心始终需要时钟以管理自身的节能功能。
- `clk_sys_busfabric`（唤醒模式下），因其阻止核心访问任何芯片寄存器，包括控制时钟使能的寄存器。
- `clk_sys_clocks`（唤醒模式下），因其阻止核心访问时钟控制寄存器。

#### 8.1.3.5.2. 系统睡眠模式

当两个核心均处于睡眠且DMA未完成事务时，系统自动进入系统睡眠模式。在系统睡眠模式下，上述时钟使能由 `WAKE_EN` 寄存器切换至 `SLEEP_EN` 寄存器。睡眠模式有助于降低芯片不活跃时时钟分配网络的功耗。如果用户未配置 `WAKE_EN` 和 `SLEEP_EN` 寄存器，系统睡眠将不会执行任何操作。

在将核心置于睡眠状态前未采取其他降低功耗措施时，单独使用系统睡眠的价值非常有限。需考虑的事项包括：

- 停止未使用的时钟源，如PLL和晶体振荡器。
- 通过增加时钟分频器来降低生成时钟的频率。
- 停止外部时钟。

为在芯片非活动时实现最大功耗节省，用户应考虑使用 DORMANT 模式（参见第 6.5.3 节，“DORMANT 状态”），该模式下时钟源由晶体振荡器和/或环形振荡器提供，且这些时钟源将被停止。

有关睡眠的更多信息，详见第 6.5.2 节，“SLEEP 状态”。

### 8.1.4. 频率计数器

频率计通过计数测试间隔内检测到的时钟边缘来测量内部及外部时钟频率。该间隔由计数 `clk_ref` 周期定义，且 `clk_ref` 必须由 XOSC 或已知频率的稳定外部信号源驱动。

用户可通过 FC0\_INTERVAL 寄存器在准确度与测试时间之间进行权衡选择。表 542，“频率计测试间隔与准确度”展示了此权衡关系：

表 542。频率计测试间隔与准确度

间隔寄存器	测试间隔	准确度
0	1μs	2048kHz
1	2μs	1024kHz
2	4μs	512kHz
3	8μs	256kHz
4	16μs	128kHz
5	32μs	64kHz
6	64μs	32kHz
7	125μs	16kHz
8	250μs	8kHz
9	500μs	4kHz
10	1ms	2kHz
11	2ms	1kHz
12	4ms	500Hz
13	8ms	250Hz
14	16ms	125Hz
15	32ms	62.5Hz

### 8.1.5. 复苏 (Resus)

可能会编写出无意中停止 `clk_sys` 的软件。这通常会导致核心及片上调试器不可恢复的锁死，用户无法追踪问题。为防止核心不可恢复锁死，提供了自动复苏电路；若在用户定义的时间间隔内检测不到信号边沿，该电路会将 `clk_sys` 切换至已知的良好时钟源 (`clk_ref`)。`clk_ref` 可由 XOSC、ROSC 或外部时钟源驱动。该时间间隔可通过 CLK\_SYS\_RESUS\_CTRL 进行编程。

#### ● 警告

若 `clk_ref` 停止，则复苏电路无法使芯片恢复正常。

启用复苏电路：

- 设置超时间隔。
- 在 CLK\_SYS\_RESUS\_CTRL 中设置 `ENABLE` 位。

检测 resus 事件的方法：

- 通过设置 INT\_E 中的中断使能位，启用 CLK\_SYS\_RESUS 中断。

- 启用 `CLOCKS_DEFAULT_IRQ` 处理器中断（参见第 3.2 节，“中断”）。

Resus 旨在作为调试辅助工具，帮助用户追踪触发 resus 的软件错误，随后修正错误并重启。可通过重新配置 `clk_sys`，随后在 `CLK_SYS_RESUS_CTRL` 中写入 `CLEAR` 位以清除 resus，从而在 resus 事件后继续运行。

### ● 警告

仅限于将 resus 用于调试。若 `clk_sys` 运行速率低于预期，可能会触发 resus。这可能导致 `clk_sys` 故障，进而损坏芯片。

## 8.1.6. 程序员模型

### 8.1.6.1. 配置时钟发生器

SDK 定义了时钟的枚举类型：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware\\_structs/include/hardware/structs/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware_structs/include/hardware/structs/clocks.h) 第 30 - 42 行

```

30 typedef enum clock_num_rp2350 {
31 clk_gpout0 = 0, //< 选择 CLK_GPOUT0 作为时钟源
32 clk_gpout1 = 1, //< 选择 CLK_GPOUT1 作为时钟源
33 clk_gpout2 = 2, //< 选择 CLK_GPOUT2 作为时钟源
34 clk_gpout3 = 3, //< 选择 CLK_GPOUT3 作为时钟源
35 clk_ref = 4, //< 选择 CLK_REF 作为时钟源
36 clk_sys = 5, //< 选择 CLK_SYS 作为时钟源
37 clk_peri = 6, //< 选择 CLK_PERI 作为时钟源
38 clk_hstx = 7, //< 选择 CLK_HSTX 作为时钟源
39 clk_usb = 8, //< 选择 CLK_USB 作为时钟源
40 clk_adc = 9, //< 选择 CLK_ADC 作为时钟源
41 CLK_COUNT
42 } clock_num_t;

```

此外，SDK 定义了一个结构体，用于描述时钟发生器的寄存器：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware\\_structs/include/hardware/structs/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware_structs/include/hardware/structs/clocks.h) 第 116 - 137 行

```

116 typedef struct {
117 _REG_(CLOCKS_CLK_GPOUT0_CTRL_OFFSET) // CLOCKS_CLK_GPOUT0_CTRL
118 // 时钟控制，除auxsrc外可动态调整
119 // 0x10000000 [28] ENABLED (0) 时钟发生器已启用
120 // 0x00100000 [20] NUDGE (0) 此信号的边沿会使输出相位发生偏移
 偏移幅度为...
121 // 0x00030000 [17:16] PHASE (0x0) 此项将使能信号延迟最多3个周期
 的...
122 // 0x00001000 [12] DC50 (0) 启用奇数除数的占空比校正，可
 被...
123 // 0x00000800 [11] 启用 (0) 干净启动和停止时钟发生器
124 // 0x00000400 [10] 终止 (0) 异步终止时钟发生器，必须启用
 被...
125 // 0x000001e0 [8:5] 辅助时钟源 (0x0) 选择辅助时钟源，切换时会出现毛刺
 when switching
126 io_rw_32 ctrl;
127
128 _REG_(CLOCKS_CLK_GPOUT0_DIV_OFFSET) // CLOCKS_CLK_GPOUT0_DIV
129 // 0xffff0000 [31:16] 整数部分 (0x0001) 时钟除数的整数部分，0 表示最大值加1，可
 被...
130 // 0x0000ffff [15:0] 分数部分 (0x0000) 除数的分数部分，可

```

```

 动态修改
131 io_rw_32 div;
132
133 _REG_(CLOCKS_CLK_GPOUT0_SELECTED_OFFSET) // CLOCKS_CLK_GPOUT0_SELECTED
134 // 指示当前选定的源 (单热编码)
135 // 0x00000001 [0] CLK_GPOUT0_SELECTED (1) 该切片无毛刺多路复用器
136 (仅限于...
137 io_ro_32 selected;
138 } clock_hw_t;

```

时钟配置需要以下信息：

- 时钟源的频率
- 时钟源的多路复用器/辅助多路复用器位置
- 目标输出频率

SDK 提供了 `clock_configure` 函数用于配置时钟：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c) 第40至133行

```

40 static void clock_configure_internal(clock_handle_t clock, uint32_t src, uint32_t auxsrc,
41 uint32_t actual_freq, uint32_t div) {
42 clock_hw_t *clock_hw = &clocks_hw->clk[clock];
43
44 // 如果增加除数，应先设置除数，再设置时钟源。否则，先设置时钟源
45 // 再设置除数。这样可以避免例如在切换
46 // 至更快的时钟源并增加除数以进行补偿时发生瞬时过速。
47 if (div > clock_hw->div)
48 clock_hw->div = div;
49
50 // 若将无毛刺时钟切片 (ref或sys) 切换至辅助源，须先从辅助源切换
51 // 以避免切换辅助多路复用器时产生毛刺。
52 // 假定 (!!!) 无毛刺源θ的速度不超过辅助源。
53 if (has_glitchless_mux(clock) && src ==
54 CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX) {
55 hw_clear_bits(&clock_hw->ctrl, CLOCKS_CLK_REF_CTRL_SRC_BITS);
56 while (!(clock_hw->selected & 1u))
57 tight_loop_contents();
58 }
59 // 如无毛刺多路复用器，则需干净地停止时钟以避免毛刺
60 // 在更改辅助多路复用器时传播。请注意，在
61 // 无毛刺时钟 (clk_sys, clk_ref) 上执行此操作将带来严重后果。
62 else {
63 // 禁用时钟。在 clk_ref 和 clk_sys 上此操作无效，
64 // 其他所有时钟的 ENABLE 位均位于相同位置。
65 hw_clear_bits(&clock_hw->ctrl, CLOCKS_CLK_GPOUT0_CTRL_ENABLE_BITS);
66 if (configured_freq[clock] > 0) {
67 // 延迟目标时钟 3 个周期，以确保 ENABLE 信号传播。
68 // 注意此处 XOSC_COUNT 无效，因为 XOSC 不一定
69 // 运行，定时器也不一定运行...
70 uint delay_cyc = configured_freq[clk_sys] / configured_freq[clock] + 1;
71 busy_wait_at_least_cycles(delay_cyc * 3);
72 }
73 }
74 // 先设置辅助多路复用器，若该时钟具备无毛刺多路复用器，则随后设置它
75 hw_write_masked(&clock_hw->ctrl,
76 (auxsrc << CLOCKS_CLK_SYS_CTRL_AUXSRC_LSB),
77 CLOCKS_CLK_SYS_CTRL_AUXSRC_BITS
78);

```

```

79 if (has_glitchless_mux(clock)) {
80 hw_write_masked(&clock_hw->ctrl,
81 src << CLOCKS_CLK_REF_CTRL_SRC_LSB,
82 CLOCKS_CLK_REF_CTRL_SRC_BITS
83);
84 while (!(clock_hw->selected & (1u << src)))
85 tight_loop_contents();
86 }
87
88 // 启用时钟。在 clk_ref 和 clk_sys 上此操作无效，
89 // 所有其他时钟的 ENABLE 位均位于相同位置。
90 hw_set_bits(&clock_hw->ctrl, CLOCKS_CLK_GPOUT0_CTRL_ENABLE_BITS);
91
92 // 源配置完成后，可确认用户提供的
93 // 分频值为安全值。
94 clock_hw->div = div;
95 configured_freq[clock] = actual_freq;
96 }
97
98 bool clock_configure(clock_handle_t clock, uint32_t src, uint32_t auxsrc, uint32_t src_freq,
99 uint32_t freq) {
100 assert(src_freq >= freq);
101
102 if (freq > src_freq)
103 return false;
104
105 uint64_t div64 = (((uint64_t) src_freq) << CLOCKS_CLK_GPOUT0_DIV_INT_LSB) / freq;
106 uint32_t div, actual_freq;
107
108 if (div64 >> 32) {
109 // 设置分频器为最大值 0
110 div = 0;
111 actual_freq = src_freq >> (32 - CLOCKS_CLK_GPOUT0_DIV_INT_LSB);
112 } else {
113 div = (uint32_t) div64;
114 actual_freq = (uint32_t) (((uint64_t) src_freq) << CLOCKS_CLK_GPOUT0_DIV_INT_LSB) /
115 div;
116 }
117
118 clock_configure_internal(clock, src, auxsrc, actual_freq, div);
119 // 存储已配置的频率
120 return true;
121
122 void clock_configure_int_divider(clock_handle_t clock, uint32_t src, uint32_t auxsrc,
123 uint32_t src_freq, uint32_t int_divider) {
124 clock_configure_internal(clock, src, auxsrc, src_freq / int_divider, int_divider <<
125 CLOCKS_CLK_GPOUT0_DIV_INT_LSB);
126 }
127
128 void clock_configure_undivided(clock_handle_t clock, uint32_t src, uint32_t auxsrc, uint32_t
129 src_freq) {
130 clock_configure_internal(clock, src, auxsrc, src_freq, 1u <<
131 CLOCKS_CLK_GPOUT0_DIV_INT_LSB);
132 }
```

`clocks_init` 调用`clock_configure`函数配置每个时钟。以下示例展示了`clk_sys`的配置：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/pico\\_runtime\\_init/runtime\\_init\\_clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/pico_runtime_init/runtime_init_clocks.c) 第100至104行

```

100 // CLK SYS = PLL SYS (通常) 125MHz / 1 = 125MHz
101 clock_configure_undivided(clk_sys,
102 CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX,
```

```
103 CLOCKS_CLK_SYS_CTRL_AUXSRC_VALUE_CLKSRC_PLL_SYS,
104 SYS_CLK_HZ);
```

一旦时钟配置完成，请调用`clock_get_hz`以获取输出频率（单位：赫兹）。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c) 第137至139行

```
137 uint32_t clock_get_hz(clock_handle_t clock) {
138 return configured_freq[clock];
139 }
```

### ● 警告

如果输入的源频率不正确，`clock_get_hz`返回的频率将不准确。

## 8.1.6.2 使用频率计数器

使用频率计数器时，程序员必须：

1. 设置参考频率：`clk_ref`。
2. 设置所测量信号的多路复用位置。参见 FC0\_SRC。
3. 等待 FC0\_STATUS 中的 `DONE` 状态位被置位。
4. 读取测量结果。

SDK 定义了`frequency_count`函数，该函数以源为参数，返回以 kHz 为单位的频率：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c) 第147行至174行

```
147 uint32_t frequency_count_khz(uint src) {
148 fc_hw_t *fc = &clocks_hw->fc0;
149
150 // 若频率计数器正在运行，则需等待其完成。即使源为NULL，该计数器仍会运行
151 // while(fc->status & CLOCKS_FC0_STATUS_RUNNING_BITS) {
152 tight_loop_contents();
153 }
154
155 // 设置参考频率
156 fc->ref_khz = clock_get_hz(clk_ref) / 1000;
157
158 // FIXME: 避免随机选择间隔，应使用最佳间隔 159 fc->interval = 10;
159
160 // 无最小值或最大值限制
161 fc->min_khz = 0;
162 fc->max_khz = 0xffffffff;
163
164 // 设置SRC，自动启动测量
165 fc->src = src;
166
167
168 while(!(fc->status & CLOCKS_FC0_STATUS_DONE_BITS)) {
169 tight_loop_contents();
170 }
171
172 // 返回结果
173 return fc->result >> CLOCKS_FC0_RESULT_KHZ_LSB;
174 }
```

还提供了一个用于将单位转换为MHz的封装函数：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/include/hardware/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/include/hardware/clocks.h) 第377行至379行

```
377 static inline float frequency_count_mhz(uint src) {
378 return ((float)(frequency_count_khz(src))) / KHZ;
379 }
```

频率计数器亦可用于测试模式。该模式允许硬件检测频率是否处于FC0\_MIN\_KHZ与FC0\_MAX\_KHZ设定的最小值与最大值之间。当 DONE位被置位时，FC0\_STATUS中的以下位之一将被设置：

- **SLOW**: 如果频率低于指定范围
- **PASS**: 如果频率在指定范围内
- **FAST**: 如果频率高于指定范围
- **DIED**: 如果时钟停止或停止运行

测试模式同样会在设置了 **DIED**、**FAST** 或 **SLOW** 时置位 **FAIL** 位。

### 8.1.6.3. 配置GPIO输出时钟

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c) 第245至263行

```
245 void clock_gpio_init_int_frac16(uint gpio, uint src, uint32_t div_int, uint16_t div_frac16)
{
246 // 注意，此处包含一个invalid_params_if，默认使用clk_gpout0
247 uint gpclk = gpio_to_gpout_clock_handle(gpio, clk_gpout0);
248
249 invalid_params_if(HARDWARE_CLOCKS, div_int >> REG_FIELD_WIDTH(
250 CLOCKS_CLK_GPOUT0_DIV_INT));
251 // 设置 gpclk 生成器
252 clocks_hw->clk[gpclk].ctrl = (src << CLOCKS_CLK_GPOUT0_CTRL_AUXSRC_LSB) |
253 CLOCKS_CLK_GPOUT0_CTRL_ENABLE_BITS;
254 #ifdef REG_FIELD_WIDTH(CLOCKS_CLK_GPOUT0_DIV_FRAC) == 16
255 clocks_hw->clk[gpclk].div = (div_int << CLOCKS_CLK_GPOUT0_DIV_INT_LSB) | (div_frac16 <<
256 CLOCKS_CLK_GPOUT0_DIV_FRAC_LSB);
257 #elif REG_FIELD_WIDTH(CLOCKS_CLK_GPOUT0_DIV_FRAC) == 8
258 clocks_hw->clk[gpclk].div = (div_int << CLOCKS_CLK_GPOUT0_DIV_INT_LSB) | ((div_frac16
259 >>8u) << CLOCKS_CLK_GPOUT0_DIV_FRAC_LSB);
260 #else
261 #error 不支持的分数位数
262 #endif
263 }
```

### 8.1.6.4. 配置GPIO输入时钟

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c) 第300至330行

```
300 bool clock_configure_gpin(clock_handle_t clock, uint gpio, uint32_t src_freq, uint32_t freq)
{
301 // 配置时钟以从GPIO输入引脚驱动
```

```

302 uint gpin = 0;
303 if (gpio == 20) gpin = 0;
304 else if (gpio == 22) gpin = 1;
305 else if (gpio == 12) gpin = 0;
306 else if (gpio == 14) gpin = 1;
307 else {
308 invalid_params_if(HARDWARE_CLOCKS, true);
309 }
310
311 // 计算信号源。GPIN 始终为 auxsrc 312 uint src = 0;

313
314 // GPIN1 == GPIN0 + 1
315 uint auxsrc = gpin0_src[clock] + gpin;
316
317 if (has_glitchless_mux(clock)) {
318 // AUX 源始终为 1
319 src = 1;
320 }
321
322 // 设置 GPIO 功能
323 gpio_set_function(gpio, GPIO_FUNC_GPCK);
324
325 // 现在已有 src、auxsrc，并已配置 gpio 输入
326 // 调用 clock_configure 以通过 gpio 运行时钟
327 return clock_configure(clock, src, auxsrc, src_freq, freq);
328 }

```

### 8.1.6.5. 启用复苏功能

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c) 第 221 至 243 行

```

221 void clocks_enable_resus(resus_callback_t resus_callback) {
222 // 如果clk_sys停止，则强制其
223 // 重启到clk_ref的默认时钟源。如果clk_ref停止运行，这将224 // 无法生效。
224
225 // 保存用户的resus回调函数
226 _resus_callback = resus_callback;
227
228 irq_set_exclusive_handler(CLOCKS_IRQ, clocks_irq_handler);
229
230 // 在clocks模块中启用resus中断
231 clocks_hw->inte = CLOCKS_INTE_CLK_SYS_RESUS_BITS;
232
233 // 启用clocks中断
234 irq_set_enabled(CLOCKS_IRQ, true);
235
236 // 2 * clk_ref频率 / clk_sys_min_freq
237 // 假设clk_ref为3MHz，且期望clk_sys不低于1MHz
238 uint timeout = 2 * 3 * 1;
239
240 // 启用具有最大超时的复苏功能
241 clocks_hw->resus.ctrl = CLOCKS_CLK_SYS_RESUS_CTRL_ENABLE_BITS | timeout;
242
243 }

```

### 8.1.6.6. 配置睡眠模式

当处理器核心和 DMA 均未请求时钟时，睡眠模式处于激活状态。例如，当 DMA 不活跃且核心0与核心1均在等待中断时，睡眠模式即处于激活状态。

`SLEEP_EN` 寄存器用于设置睡眠模式下运行的时钟。hello\_sleep 示例（hello\_sleep\_aon.c，位于 pico-playground GitHub 仓库）演示了如何使芯片进入睡眠，直到 AON 定时器触发。

#### 注意

`clk_sys` 在睡眠模式期间始终发送至 `proc0` 和 `proc1`，因为处理器必须为部分逻辑提供时钟，以便醒来。

Pico Extras: [https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\\_common/pico\\_sleep/sleep.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/pico_sleep/sleep.c) 第159至183行

```

159 void sleep_goto_sleep_until(struct timespec *ts, aon_timer_alarm_handler_t callback)
160 {
161
162 // 我们应已调用 sleep_run_from_dormant_source 函数
163 // 该函数仅在休眠模式下需要，尽管它节省了休眠时使用 xosc 运行的功耗
164
165 //assert(dormant_source_valid(_dormant_source));
166
167 clocks_hw->sleep_en0 = CLOCKS_SLEEP_EN0_CLK_REF_POWMAN_BITS;
168 clocks_hw->sleep_en1 = 0x0;
169
170 aon_timer_enable_alarm(ts, callback, false);
171
172 stdio_flush();
173
174 // 启用处理器深度睡眠
175 processor_deep_sleep();
176
177 // 进入睡眠状态
178 __wfi();
179 }
```

### 8.1.7. 寄存器列表

时钟寄存器的起始地址为 `0x40010000`（在 SDK 中定义为 `CLOCKS_BASE`）。

表543. CLOCKS 寄存器列表

偏移量	名称	说明
0x00	<code>CLK_GPOUT0_CTRL</code>	时钟控制，可动态更改（auxsrc除外）
0x04	<code>CLK_GPOUT0_DIV</code>	
0x08	<code>CLK_GPOUT0_SELECTED</code>	指示当前选定的src（单热编码）
0x0c	<code>CLK_GPOUT1_CTRL</code>	时钟控制，可动态更改（auxsrc除外）
0x10	<code>CLK_GPOUT1_DIV</code>	
0x14	<code>CLK_GPOUT1_SELECTED</code>	指示当前选定的src（单热编码）
0x18	<code>CLK_GPOUT2_CTRL</code>	时钟控制，可动态更改（auxsrc除外）
0x1c	<code>CLK_GPOUT2_DIV</code>	
0x20	<code>CLK_GPOUT2_SELECTED</code>	指示当前选定的src（单热编码）
0x24	<code>CLK_GPOUT3_CTRL</code>	时钟控制，可动态更改（auxsrc除外）

偏移量	名称	说明
0x28	CLK_GPOUT3_DIV	
0x2c	CLK_GPOUT3_SELECTED	指示当前选定的src（单热编码）
0x30	CLK_REF_CTRL	时钟控制，可动态更改（auxsrc除外）
0x34	CLK_REF_DIV	
0x38	CLK_REF_SELECTED	指示当前选定的src（单热编码）
0x3c	CLK_SYS_CTRL	时钟控制，可动态更改（auxsrc除外）
0x40	CLK_SYS_DIV	
0x44	CLK_SYS_SELECTED	指示当前选定的src（单热编码）
0x48	CLK_PERI_CTRL	时钟控制，可动态更改（auxsrc除外）
0x4c	CLK_PERI_DIV	
0x50	CLK_PERI_SELECTED	指示当前选定的src（单热编码）
0x54	CLK_HSTX_CTRL	时钟控制，可动态更改（auxsrc除外）
0x58	CLK_HSTX_DIV	
0x5c	CLK_HSTX_SELECTED	指示当前选定的src（单热编码）
0x60	CLK_USB_CTRL	时钟控制，可动态更改（auxsrc除外）
0x64	CLK_USB_DIV	
0x68	CLK_USB_SELECTED	指示当前选定的src（单热编码）
0x6c	CLK_ADC_CTRL	时钟控制，可动态更改（auxsrc除外）
0x70	CLK_ADC_DIV	
0x74	CLK_ADC_SELECTED	指示当前选定的src（单热编码）
0x78	DFTCLK_XOSC_CTRL	
0x7c	DFTCLK_ROSC_CTRL	
0x80	DFTCLK_LPOS_C_CTRL	
0x84	CLK_SYS_RESUS_CTRL	
0x88	CLK_SYS_RESUS_STATUS	
0x8c	FC0_REF_KHZ	参考时钟频率，单位为kHz
0x90	FC0_MIN_KHZ	最小通过频率，单位为kHz。此项为可选项。如不使用通过/失败标志，设为0
0x94	FC0_MAX_KHZ	最大通过频率，单位为kHz。此项为可选项。如不使用通过/失败标志，设为0xffffffff
0x98	FC0_DELAY	延迟频率计数开始以允许多路复用器稳定 延迟以参考时钟周期倍数计量
0x9c	FC0_INTERVAL	测试间隔为0.98微秒 × 2**interval，但可视为1微秒 × 2**interval 默认测试间隔为250微秒
0xa0	FC0_SRC	发送至频率计数器的时钟，不需要时设为0 写入此寄存器以启动频率计数
0xa4	FC0_STATUS	频率计数器状态

偏移量	名称	说明
0xa8	FC0_RESULT	频率测量结果，仅在status_done=1时有效
0xac	WAKE_EN0	在唤醒模式下启用时钟
0xb0	WAKE_EN1	在唤醒模式下启用时钟
0xb4	SLEEP_EN0	在睡眠模式下启用时钟
0xb8	SLEEP_EN1	在睡眠模式下启用时钟
0xbc	启用0	指示时钟启用状态
0xc0	启用1	指示时钟启用状态
0xc4	中断	原始中断
0xc8	INTE	中断使能
0xcc	INTF	中断触发
0xd0	INTS	掩码与强制后的中断状态

## 时钟：CLK\_GPOUT0\_CTRL 寄存器

偏移: 0x00

### 说明

时钟控制，可动态更改（auxsrc除外）

表 544。  
CLK\_GPOUT0\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0
27:21	保留。	-	-
20	<b>NUDGE</b> : 该信号的边沿将输出相位移动一个输入时钟周期  此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位：该字段可延迟使能信号，最长达输入时钟的3个周期 本字段必须在启用时钟前设置，方可生效	读写	0x0
15:13	保留。	-	-
12	<b>DC50</b> : 启用奇数分频的占空比校正，支持动态调整	读写	0x0
11	启用：干净地启动或停止时钟发生器	读写	0x0
10	终止：异步终止时钟发生器，取消终止前必须先将启用信号置低	读写	0x0
9	保留。	-	-
8:5	辅助源：选择辅助时钟源，切换时将产生毛刺	读写	0x0
	枚举值：		
	0x0 → CLKSRC_PLL_SYS		
	0x1 → CLKSRC_GPIO0		

位	描述	类型	复位
	0x2 → CLKSRC_GPIN1		
	0x3 → CLKSRC_PLL_USB		
	0x4 → CLKSRC_PLL_USB_PRIMARY_REF_OPCG		
	0x5 → ROSC_CLKSRC		
	0x6 → XOSC_CLKSRC		
	0x7 → LPOSC_CLKSRC		
	0x8 → CLK_SYS		
	0x9 → CLK_USB		
	0xa → CLK_ADC		
	0xb → CLK_REF		
	0xc → CLK_PERI		
	0xd → CLK_HSTX		
	0xe → OTP_CLK2FC		
4:0	保留。	-	-

## CLOCKS: CLK\_GPOUT0\_DIV 寄存器

偏移: 0x04

表 545。  
CLK\_GPOUT0\_DIV  
寄存器

位	描述	类型	复位
31:16	<b>INT</b> : 时钟除数的整数部分，范围 0 → max+1，可动态调整	读写	0x0001
15:0	<b>FRAC</b> : 除数的小数部分，可动态调整	读写	0x0000

## CLOCKS: CLK\_GPOUT0\_SELECTED 寄存器

偏移: 0x08

### 说明

指示当前选定的src (单热编码)

表 546。  
CLK\_GPOUT0\_SELECT  
ED 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## CLOCKS: CLK\_GPOUT1\_CTRL 寄存器

偏移: 0x0c

### 描述

时钟控制，可动态更改 (auxsrc除外)

表 547。  
CLK\_GPOUT1\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0

位	描述	类型	复位
27:21	保留。	-	-
20	<b>NUDGE</b> : 该信号的边沿将输出相位移动一个输入时钟周期 此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位: 该字段可延迟使能信号, 最长达输入时钟的3个周期 本字段必须在启用时钟前设置, 方可生效	读写	0x0
15:13	保留。	-	-
12	<b>DC50</b> : 启用奇数分频的占空比校正, 支持动态调整	读写	0x0
11	启用: 干净地启动或停止时钟发生器	读写	0x0
10	终止: 异步终止时钟发生器, 取消终止前必须先将启用信号置低	读写	0x0
9	保留。	-	-
8:5	辅助源: 选择辅助时钟源, 切换时将产生毛刺 枚举值:	读写	0x0
	0x0 → CLKSRC_PLL_SYS		
	0x1 → CLKSRC_GPIN0		
	0x2 → CLKSRC_GPIN1		
	0x3 → CLKSRC_PLL_USB		
	0x4 → CLKSRC_PLL_USB_PRIMARY_REF_OPCG		
	0x5 → ROSC_CLKSRC		
	0x6 → XOSC_CLKSRC		
	0x7 → LPOSC_CLKSRC		
	0x8 → CLK_SYS		
	0x9 → CLK_USB		
	0xa → CLK_ADC		
	0xb → CLK_REF		
	0xc → CLK_PERI		
	0xd → CLK_HSTX		
	0xe → OTP_CLK2FC		
4:0	保留。	-	-

## CLOCKS: CLK\_GPOUT1\_DIV 寄存器

偏移: 0x10

表 548。  
CLK\_GPOUT1\_DIV  
寄存器

位	描述	类型	复位
31:16	<b>INT</b> : 时钟除数的整数部分, 范围 0 → max+1, 可动态调整	读写	0x0001

位	描述	类型	复位
15:0	<b>FRAC</b> : 除数的小数部分，可动态调整	读写	0x0000

## CLOCKS: CLK\_GPOUT1\_SELECTED 寄存器

偏移: 0x14

### 说明

指示当前选定的src (单热编码)

表 549。  
CLK\_GPOUT1\_SELECT  
ED 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟: CLK\_GPOUT2\_CTRL 寄存器

偏移量: 0x18

### 说明

时钟控制，可动态更改 (auxsrc除外)

表 550  
CLK\_GPOUT2\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用: 时钟发生器已启用	只读	0x0
27:21	保留。	-	-
20	<b>NUDGE</b> : 该信号的边沿将输出相位移动一个输入时钟周期  此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位: 该字段可延迟使能信号，最长达输入时钟的3个周期 本字段必须在启用时钟前设置，方可生效	读写	0x0
15:13	保留。	-	-
12	<b>DC50</b> : 启用奇数分频的占空比校正，支持动态调整	读写	0x0
11	启用: 干净地启动或停止时钟发生器	读写	0x0
10	终止: 异步终止时钟发生器，取消终止前必须先将启用信号置低	读写	0x0
9	保留。	-	-
8:5	辅助源: 选择辅助时钟源，切换时将产生毛刺  枚举值:  0x0 → CLKSRC_PLL_SYS 0x1 → CLKSRC_GPIO 0x2 → CLKSRC_GPIO1 0x3 → CLKSRC_PLL_USB 0x4 → CLKSRC_PLL_USB_PRIMARY_REF_OPCG	读写	0x0

位	描述	类型	复位
	0x5 → ROSC_CLKSRC_PH		
	0x6 → XOSC_CLKSRC		
	0x7 → LPOSC_CLKSRC		
	0x8 → CLK_SYS		
	0x9 → CLK_USB		
	0xa → CLK_ADC		
	0xb → CLK_REF		
	0xc → CLK_PERI		
	0xd → CLK_HSTX		
	0xe → OTP_CLK2FC		
4:0	保留。	-	-

## 时钟：CLK\_GPOUT2\_DIV 寄存器

偏移：0x1c

表 551  
CLK\_GPOUT2\_DIV  
寄存器

位	描述	类型	复位
31:16	<b>INT</b> : 时钟除数的整数部分，范围 0 → max+1，可动态调整	读写	0x0001
15:0	<b>FRAC</b> : 除数的小数部分，可动态调整	读写	0x0000

## 时钟：CLK\_GPOUT2\_SELECTED 寄存器

偏移：0x20

### 说明

指示当前选定的src（单热编码）

表 552  
CLK\_GPOUT2\_SELECTED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟：CLK\_GPOUT3\_CTRL 寄存器

偏移量：0x24

### 说明

时钟控制，可动态更改（auxsrc除外）

表 553  
CLK\_GPOUT3\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0
27:21	保留。	-	-

位	描述	类型	复位
20	<b>NUDGE</b> : 该信号的边沿将输出相位移动一个输入时钟周期 此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位: 该字段可延迟使能信号, 最长达输入时钟的3个周期 本字段必须在启用时钟前设置, 方可生效	读写	0x0
15:13	保留。	-	-
12	<b>DC50</b> : 启用奇数分频的占空比校正, 支持动态调整	读写	0x0
11	启用: 干净地启动或停止时钟发生器	读写	0x0
10	终止: 异步终止时钟发生器, 取消终止前必须先将启用信号置低	读写	0x0
9	保留。	-	-
8:5	辅助源: 选择辅助时钟源, 切换时将产生毛刺 枚举值:	读写	0x0
	0x0 → CLKSRC_PLL_SYS		
	0x1 → CLKSRC_GPIN0		
	0x2 → CLKSRC_GPIN1		
	0x3 → CLKSRC_PLL_USB		
	0x4 → CLKSRC_PLL_USB_PRIMARY_REF_OPCG		
	0x5 → ROSC_CLKSRC_PH		
	0x6 → XOSC_CLKSRC		
	0x7 → LPOSC_CLKSRC		
	0x8 → CLK_SYS		
	0x9 → CLK_USB		
	0xa → CLK_ADC		
	0xb → CLK_REF		
	0xc → CLK_PERI		
	0xd → CLK_HSTX		
	0xe → OTP_CLK2FC		
4:0	保留。	-	-

## 时钟: CLK\_GPOUT3\_DIV 寄存器

偏移量: 0x28

表 554  
CLK\_GPOUT3\_DIV  
寄存器

位	描述	类型	复位
31:16	<b>INT</b> : 时钟除数的整数部分，范围 $0 \rightarrow \text{max}+1$ ，可动态调整	读写	0x0001
15:0	<b>FRAC</b> : 除数的小数部分，可动态调整	读写	0x0000

## 时钟：CLK\_GPOUT3\_SELECTED 寄存器

偏移：0x2c

### 说明

指示当前选定的src（单热编码）

表555。  
CLK\_GPOUT3\_SELECTED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟：CLK\_REF\_CTRL 寄存器

偏移：0x30

### 描述

时钟控制，可动态更改（auxsrc除外）

表556。  
CLK\_REF\_CTRL  
寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:5	辅助源：选择辅助时钟源，切换时将产生毛刺	读写	0x0
	枚举值：		
	0x0 → CLKSRC_PLL_USB		
	0x1 → CLKSRC_GPIO		
	0x2 → CLKSRC_GPIO1		
	0x3 → CLKSRC_PLL_USB_PRIMARY_REF_OPCG		
4:2	保留。	-	-
1:0	<b>SRC</b> ：选择无毛刺时钟源，可动态更改	读写	-
	枚举值：		
	0x0 → ROSC_CLKSRC_PH		
	0x1 → CLKSRC_CLK_REF_AUX		
	0x2 → XOSC_CLKSRC		
	0x3 → LPOSCLKSRC		

## 时钟：CLK\_REF\_DIV 寄存器

偏移量：0x34

表557。  
CLK\_REF\_DIV 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>INT</b> : 时钟除数的整数部分，范围 $0 \rightarrow \text{max}+1$ ，可动态调整	读写	0x01

位	描述	类型	复位
15:0	保留。	-	-

## 时钟：CLK\_REF\_SELECTED 寄存器

偏移: 0x38

### 描述

指示当前选定的src（单热编码）

表 558。  
CLK\_REF\_SELECTED  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	无毛刺多路复用器不会瞬时切换（以避免毛刺），因此软件应轮询该寄存器以等待切换完成。该寄存器包含 CTRL_SRC 字段枚举的每个时钟源对应的一个译码位。任一时间最多只有一个该类位被置位，表示该时钟当前出现在无毛刺多路复用器的输出端。在切换过程中，该寄存器可能短暂显示全零。	只读	0x1

## 时钟：CLK\_SYS\_CTRL 寄存器

偏移: 0x3c

### 描述

时钟控制，可动态更改（auxsrc除外）

表 559。  
CLK\_SYS\_CTRL  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:5	辅助源：选择辅助时钟源，切换时将产生毛刺	读写	0x2
	枚举值：		
	0x0 → CLK_SRC_PLL_SYS		
	0x1 → CLK_SRC_PLL_USB		
	0x2 → ROSC_CLKSRC		
	0x3 → XOSC_CLKSRC		
	0x4 → CLK_SRC_GPIO		
	0x5 → CLK_SRC_GPIO1		
4:1	保留。	-	-
0	<b>SRC</b> ：选择无毛刺时钟源，可动态更改	读写	0x1
	枚举值：		
	0x0 → CLK_REF		
	0x1 → CLK_SRC_CLK_SYS_AUX		

## CLOCKS：CLK\_SYS\_DIV 寄存器

偏移量：0x40

表 560。  
CLK\_SYS\_DIV 寄存器

位	描述	类型	复位
31:16	<b>INT</b> : 时钟除数的整数部分，范围 $0 \rightarrow \text{max}+1$ ，可动态调整	读写	0x0001
15:0	<b>FRAC</b> : 除数的小数部分，可动态调整	读写	0x0000

## CLOCKS: CLK\_SYS\_SELECTED 寄存器

偏移: 0x44

### 说明

指示当前选定的src (单热编码)

表 561。  
CLK\_SYS\_SELECTED  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1:0	无毛刺多路复用器不会瞬时切换（以避免毛刺），因此软件应轮询该寄存器以等待切换完成。该寄存器包含 CTRL_SRC 字段枚举的每个时钟源对应的一个译码位。任一时间最多只有一个该类位被置位，表示该时钟当前出现在无毛刺多路复用器的输出端。在切换过程中，该寄存器可能短暂显示全零。	只读	0x1

## CLOCKS: CLK\_PERI\_CTRL 寄存器

偏移: 0x48

### 说明

时钟控制，可动态更改 (auxsrc除外)

表 562。  
CLK\_PERI\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0
27:12	保留。	-	-
11	启用：干净地启动或停止时钟发生器	读写	0x0
10	终止：异步终止时钟发生器，取消终止前必须先将启用信号置低	读写	0x0
9:8	保留。	-	-
7:5	辅助源：选择辅助时钟源，切换时将产生毛刺	读写	0x0
	枚举值：		
	0x0 → CLK_SYS		
	0x1 → CLKSRC_PLL_SYS		
	0x2 → CLKSRC_PLL_USB		
	0x3 → ROSC_CLKSRC_PH		
	0x4 → XOSC_CLKSRC		
	0x5 → CLKSRC_GPIN0		
	0x6 → CLKSRC_GPIN1		
4:0	保留。	-	-

## 时钟 (CLOCKS) : CLK\_PERI\_DIV 寄存器

偏移量: 0x4c

表 563。  
CLK\_PERI\_DIV  
寄存器

位	描述	类型	复位
31:18	保留。	-	-
17:16	<b>INT:</b> 时钟除数的整数部分，范围 0 → max+1，可动态调整	读写	0x1
15:0	保留。	-	-

## 时钟 (CLOCKS) : CLK\_PERI\_SELECTED 寄存器

偏移: 0x50

### 说明

指示当前选定的src (单热编码)

表 564。  
CLK\_PERI\_SELECTED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟 (CLOCKS) : CLK\_HSTX\_CTRL 寄存器

偏移: 0x54

### 说明

时钟控制，可动态更改 (auxsrc除外)

表 565。  
CLK\_HSTX\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0
27:21	保留。	-	-
20	<b>NUDGE:</b> 该信号的边沿将输出相位移动一个输入时钟周期  此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位：该字段可延迟使能信号，最长达输入时钟的3个周期 本字段必须在启用时钟前设置，方可生效	读写	0x0
15:12	保留。	-	-
11	启用：干净地启动或停止时钟发生器	读写	0x0
10	终止：异步终止时钟发生器，取消终止前必须先将启用信号置低	读写	0x0
9:8	保留。	-	-
7:5	辅助源：选择辅助时钟源，切换时将产生毛刺	读写	0x0
	枚举值：		
	0x0 → CLK_SYS		
	0x1 → CLKSRC_PLL_SYS		

位	描述	类型	复位
	0x2 → CLKSRC_PLL_USB		
	0x3 → CLKSRC_GPIN0		
	0x4 → CLKSRC_GPIN1		
4:0	保留。	-	-

## 时钟 (CLOCKS) : CLK\_HSTX\_DIV 寄存器

偏移量: 0x58

表 566。  
CLK\_HSTX\_DIV  
寄存器

位	描述	类型	复位
31:18	保留。	-	-
17:16	<b>INT</b> : 时钟除数的整数部分，范围 0 → max+1，可动态调整	读写	0x1
15:0	保留。	-	-

## 时钟 (CLOCKS) : CLK\_HSTX\_SELECTED 寄存器

偏移: 0x5c

### 描述

指示当前选定的src (单热编码)

表 567。  
CLK\_HSTX\_SELECTED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟: CLK\_USB\_CTRL 寄存器

偏移: 0x60

### 描述

时钟控制，可动态更改 (auxsrc除外)

表 568。  
CLK\_USB\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0
27:21	保留。	-	-
20	<b>NUDGE</b> : 该信号的边沿将输出相位移动一个输入时钟周期  此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位：该字段可延迟使能信号，最长达输入时钟的3个周期 本字段必须在启用时钟前设置，方可生效	读写	0x0
15:12	保留。	-	-
11	启用：干净地启动或停止时钟发生器	读写	0x0

位	描述	类型	复位
10	终止：异步终止时钟发生器，取消终止前必须先将启用信号置低	读写	0x0
9:8	保留。	-	-
7:5	辅助源：选择辅助时钟源，切换时将产生毛刺	读写	0x0
	枚举值：		
	0x0 → CLKSRC_PLL_USB		
	0x1 → CLKSRC_PLL_SYS		
	0x2 → ROOSC_CLKSRC_PH		
	0x3 → XOSC_CLKSRC		
	0x4 → CLKSRC_GPIO		
	0x5 → CLKSRC_GPIO1		
4:0	保留。	-	-

## 时钟：CLK\_USB\_DIV 寄存器

偏移: 0x64

表 569。  
CLK\_USB\_DIV 寄存器

位	描述	类型	复位
31:20	保留。	-	-
19:16	INT：时钟除数的整数部分，范围 0 → max+1，可动态调整	读写	0x1
15:0	保留。	-	-

## 时钟：CLK\_USB\_SELECTED 寄存器

偏移: 0x68

### 描述

指示当前选定的src（单热编码）

表 570。  
CLK\_USB\_SELECTED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟：CLK\_ADC\_CTRL 寄存器

偏移: 0x6c

### 描述

时钟控制，可动态更改（auxsrc除外）

表 571。  
CLK\_ADC\_CTRL  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	启用：时钟发生器已启用	只读	0x0
27:21	保留。	-	-

位	描述	类型	复位
20	<b>NUDGE</b> : 该信号的边沿将输出相位移动一个输入时钟周期 此操作可在任何时间执行	读写	0x0
19:18	保留。	-	-
17:16	相位: 该字段可延迟使能信号, 最长达输入时钟的3个周期 本字段必须在启用时钟前设置, 方可生效	读写	0x0
15:12	保留。	-	-
11	启用: 干净地启动或停止时钟发生器	读写	0x0
10	终止: 异步终止时钟发生器, 取消终止前必须先将启用信号置低	读写	0x0
9:8	保留。	-	-
7:5	辅助源: 选择辅助时钟源, 切换时将产生毛刺 枚举值: 0x0 → CLKSRC_PLL_USB 0x1 → CLKSRC_PLL_SYS 0x2 → ROSC_CLKSRC_PH 0x3 → XOSC_CLKSRC 0x4 → CLKSRC_GPIN0 0x5 → CLKSRC_GPIN1	读写	0x0
4:0	保留。	-	-

## 时钟: CLK\_ADC\_DIV 寄存器

偏移量: 0x70

表 572。  
CLK\_ADC\_DIV 寄存器

位	描述	类型	复位
31:20	保留。	-	-
19:16	<b>INT</b> : 时钟除数的整数部分, 范围 0 → max+1, 可动态调整	读写	0x1
15:0	保留。	-	-

## 时钟: CLK\_ADC\_SELECTED 寄存器

偏移: 0x74

### 描述

指示当前选定的src (单热编码)

表 573。  
CLK\_ADC\_SELECTED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	该切片不具备无毛刺多路复用器（仅包含 AUX_SRC 字段，未包含 SRC 字段），因此该寄存器硬编码为 0x1。	只读	0x1

## 时钟：DFTCLK\_XOSC\_CTRL 寄存器

偏移量: 0x78

表 574。  
DFTCLK\_XOSC\_CTRL  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1:0	源	读写	0x0
	枚举值：		
	0x0 → 无		
	0x1 → CLKSRC_PLL_USB_PRIMARY		
	0x2 → CLKSRC_GPIN0		

## 时钟：DFTCLK\_ROSC\_CTRL 寄存器

偏移: 0x7c

表 575。  
DFTCLK\_ROSC\_CTRL  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1:0	源	读写	0x0
	枚举值：		
	0x0 → 无		
	0x1 → CLKSRC_PLL_SYS_PRIMARY_ROSC		
	0x2 → CLKSRC_GPIN1		

## CLOCKS: DFTCLK\_LPOSC\_CTRL 寄存器

偏移: 0x80

表 576。  
DFTCLK\_LPOSC\_CTRL  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1:0	源	读写	0x0
	枚举值：		
	0x0 → 无		
	0x1 → CLKSRC_PLL_USB_PRIMARY_LPOSC		
	0x2 → CLKSRC_GPIN1		

## CLOCKS: CLK\_SYS\_RESUS\_CTRL 寄存器

偏移: 0x84

表 577。  
CLK\_SYS\_RESUS\_CTR  
L 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>CLEAR</b> : 用于故障修正后清除复位	读写	0x0
15:13	保留。	-	-
12	<b>FRCE</b> : 强制复位, 仅限测试用途	读写	0x0
11:9	保留。	-	-
8	<b>ENABLE</b> : 启用复位	读写	0x0
7:0	<b>TIMEOUT</b> : 以clk_ref周期计, 且须 $\geq 2 \times \text{clk\_ref\_freq}/\text{min\_clk\_tst\_freq}$	读写	0xff

## CLOCKS: CLK\_SYS\_RESUS\_STATUS 寄存器

偏移: 0x88

表 578。  
CLK\_SYS\_RESUS\_STA  
TUS 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>RESUSSED</b> : 时钟已复位, 需纠正错误后发送 ctrl_clear=1	只读	0x0

## CLOCKS: FC0\_REF\_KHZ 寄存器

偏移量: 0x8c

表 579。  
FC0\_REF\_KHZ 寄存器

位	描述	类型	复位
31:20	保留。	-	-
19:0	参考时钟频率, 单位为kHz	读写	0x000000

## CLOCKS: FC0\_MIN\_KHZ 寄存器

偏移量: 0x90

表 580。  
FC0\_MIN\_KHZ  
寄存器

位	描述	类型	复位
31:25	保留。	-	-
24:0	最小通过频率, 单位为kHz。此项为可选项。如不使用通过/失败标志, 设为0	读写	0x00000000

## CLOCKS: FC0\_MAX\_KHZ 寄存器

偏移: 0x94

表 581。  
FC0\_MAX\_KHZ  
寄存器

位	描述	类型	复位
31:25	保留。	-	-
24:0	最大通过频率, 单位为kHz。此项为可选项。如不使用通过/失败标志, 设为0x1fffffff	读写	0x1fffffff

## CLOCKS: FC0\_DELAY 寄存器

偏移: 0x98

表582。FC0\_DELAY 寄存器

位	描述	类型	复位
31:3	保留。	-	-
2:0	延迟频率计数开始以允许多路复用器稳定 延迟以参考时钟周期倍数计量	读写	0x1

## CLOCKS: FC0\_INTERVAL 寄存器

偏移: 0x9c

表583。  
FC0\_INTERVAL 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	测试间隔为 $0.98\text{微秒} \times 2^{\text{interval}}$ , 但可视为 $1\text{微秒} \times 2^{\text{interval}}$ 默认测试间隔为250微秒	读写	0x8

## CLOCKS: FC0\_SRC 寄存器

偏移: 0xa0

表584。FC0\_SRC 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	发送至频率计数器的时钟, 不需要时设为0 写入此寄存器以启动频率计数	读写	0x00
	枚举值:		
0x00	→ NULL		
0x01	→ PLL_SYS_CLKSRC_PRIMARY		
0x02	→ PLL_USB_CLKSRC_PRIMARY		
0x03	→ ROSC_CLKSRC		
0x04	→ ROSC_CLKSRC_PH		
0x05	→ XOSC_CLKSRC		
0x06	→ CLKSRC_GPIN0		
0x07	→ CLKSRC_GPIN1		
0x08	→ CLK_REF		
0x09	→ CLK_SYS		
0x0a	→ CLK_PERI		
0x0b	→ CLK_USB		
0x0c	→ CLK_ADC		
0x0d	→ CLK_HSTX		
0x0e	→ LPOSOC_CLKSRC		
0x0f	→ OTP_CLK2FC		
0x10	→ PLL_USB_CLKSRC_PRIMARY_DFT		

## CLOCKS: FC0\_STATUS 寄存器

偏移: 0xa4

**描述**

频率计数器状态

表 585.  
FC0\_STATUS 寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<b>DIED</b> : 测试时钟在测试过程中停止	只读	0x0
27:25	保留。	-	-
24	<b>FAST</b> : 测试时钟快于预期, 仅当 status_done=1 时有效	只读	0x0
23:21	保留。	-	-
20	<b>SLOW</b> : 测试时钟慢于预期, 仅当 status_done=1 时有效	只读	0x0
19:17	保留。	-	-
16	<b>FAIL</b> : 测试失败	只读	0x0
15:13	保留。	-	-
12	<b>WAITING</b> : 等待测试时钟启动	只读	0x0
11:9	保留。	-	-
8	<b>RUNNING</b> : 测试进行中	只读	0x0
7:5	保留。	-	-
4	<b>DONE</b> : 测试完成	只读	0x0
3:1	保留。	-	-
0	<b>PASS</b> : 测试通过	只读	0x0

**CLOCKS: FC0\_RESULT 寄存器**

偏移: 0xa8

**描述**

频率测量结果, 仅当 status\_done=1 时有效

表 586.  
FC0\_RESULT 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:5	<b>KHZ</b>	只读	0x0000000
4:0	<b>FRAC</b>	只读	0x00

**CLOCKS: WAKE\_EN0 寄存器**

偏移: 0xac

**描述**

在唤醒模式下启用时钟

表 587. WAKE\_EN0  
寄存器

位	描述	类型	复位
31	<b>CLK_SYS_SIO</b>	读写	0x1
30	<b>CLK_SYS_SHA256</b>	读写	0x1
29	<b>CLK_SYS_PSM</b>	读写	0x1
28	<b>CLK_SYS_ROSC</b>	读写	0x1

位	描述	类型	复位
27	<b>CLK_SYS_ROM</b>	读写	0x1
26	<b>CLK_SYS_RESETS</b>	读写	0x1
25	<b>CLK_SYS_PWM</b>	读写	0x1
24	<b>CLK_SYS_POWMAN</b>	读写	0x1
23	<b>CLK_REF_POWMAN</b>	读写	0x1
22	<b>CLK_SYS_PLL_USB</b>	读写	0x1
21	<b>CLK_SYS_PLL_SYS</b>	读写	0x1
20	<b>CLK_SYS_PIO2</b>	读写	0x1
19	<b>CLK_SYS_PIO1</b>	读写	0x1
18	<b>CLK_SYS_PIO0</b>	读写	0x1
17	<b>CLK_SYS_PADS</b>	读写	0x1
16	<b>CLK_SYS OTP</b>	读写	0x1
15	<b>CLK_REF OTP</b>	读写	0x1
14	<b>CLK_SYS_JTAG</b>	读写	0x1
13	<b>CLK_SYS_IO</b>	读写	0x1
12	<b>CLK_SYS_I2C1</b>	读写	0x1
11	<b>CLK_SYS_I2C0</b>	读写	0x1
10	<b>CLK_SYS_HSTX</b>	读写	0x1
9	<b>CLK_HSTX</b>	读写	0x1
8	<b>CLK_SYS_GLITCH_DETECTOR</b>	读写	0x1
7	<b>CLK_SYS_DMA</b>	读写	0x1
6	<b>CLK_SYS_BUSFABRIC</b>	读写	0x1
5	<b>CLK_SYS_BUSCTRL</b>	读写	0x1
4	<b>CLK_SYS_BOOTRAM</b>	读写	0x1
3	<b>CLK_SYS_ADC</b>	读写	0x1
2	<b>CLK_ADC_ADC</b>	读写	0x1
1	<b>CLK_SYS_ACCESSCTRL</b>	读写	0x1
0	<b>CLK_SYS_CLOCKS</b>	读写	0x1

## CLOCKS: WAKE\_EN1 寄存器

偏移: 0xb0

### 说明

在唤醒模式下启用时钟

表 588. WAKE\_EN1  
寄存器

位	描述	类型	复位
31	保留。	-	-
30	<b>CLK_SYS_XOSC</b>	读写	0x1

位	描述	类型	复位
29	<b>CLK_SYS_XIP</b>	读写	0x1
28	<b>CLK_SYS_WATCHDOG</b>	读写	0x1
27	<b>CLK_USB</b>	读写	0x1
26	<b>CLK_SYS_USBCTRL</b>	读写	0x1
25	<b>CLK_SYS_UART1</b>	读写	0x1
24	<b>CLK_PERI_UART1</b>	读写	0x1
23	<b>CLK_SYS_UART0</b>	读写	0x1
22	<b>CLK_PERI_UART0</b>	读写	0x1
21	<b>CLK_SYS_TRNG</b>	读写	0x1
20	<b>CLK_SYS_TIMER1</b>	读写	0x1
19	<b>CLK_SYS_TIMER0</b>	读写	0x1
18	<b>CLK_SYS_TICKS</b>	读写	0x1
17	<b>CLK_REF_TICKS</b>	读写	0x1
16	<b>CLK_SYS_TBMAN</b>	读写	0x1
15	<b>CLK_SYS_SYSINFO</b>	读写	0x1
14	<b>CLK_SYS_SYSCFG</b>	读写	0x1
13	<b>CLK_SYS_SRAM9</b>	读写	0x1
12	<b>CLK_SYS_SRAM8</b>	读写	0x1
11	<b>CLK_SYS_SRAM7</b>	读写	0x1
10	<b>CLK_SYS_SRAM6</b>	读写	0x1
9	<b>CLK_SYS_SRAM5</b>	读写	0x1
8	<b>CLK_SYS_SRAM4</b>	读写	0x1
7	<b>CLK_SYS_SRAM3</b>	读写	0x1
6	<b>CLK_SYS_SRAM2</b>	读写	0x1
5	<b>CLK_SYS_SRAM1</b>	读写	0x1
4	<b>CLK_SYS_SRAM0</b>	读写	0x1
3	<b>CLK_SYS_SPI1</b>	读写	0x1
2	<b>CLK_PERI_SPI1</b>	读写	0x1
1	<b>CLK_SYS_SPI0</b>	读写	0x1
0	<b>CLK_PERI_SPI0</b>	读写	0x1

## CLOCKS: SLEEP\_EN0 寄存器

偏移: 0xb4

### 说明

在睡眠模式下启用时钟

表 589. SLEEP\_EN0  
寄存器

位	描述	类型	复位
31	<b>CLK_SYS_SIO</b>	读写	0x1
30	<b>CLK_SYS_SHA256</b>	读写	0x1
29	<b>CLK_SYS_PSM</b>	读写	0x1
28	<b>CLK_SYS_ROSC</b>	读写	0x1
27	<b>CLK_SYS_ROM</b>	读写	0x1
26	<b>CLK_SYS_RESETS</b>	读写	0x1
25	<b>CLK_SYS_PWM</b>	读写	0x1
24	<b>CLK_SYS_POWMAN</b>	读写	0x1
23	<b>CLK_REF_POWMAN</b>	读写	0x1
22	<b>CLK_SYS_PLL_USB</b>	读写	0x1
21	<b>CLK_SYS_PLL_SYS</b>	读写	0x1
20	<b>CLK_SYS_PIO2</b>	读写	0x1
19	<b>CLK_SYS_PIO1</b>	读写	0x1
18	<b>CLK_SYS_PIO0</b>	读写	0x1
17	<b>CLK_SYS_PADS</b>	读写	0x1
16	<b>CLK_SYS OTP</b>	读写	0x1
15	<b>CLK_REF OTP</b>	读写	0x1
14	<b>CLK_SYS_JTAG</b>	读写	0x1
13	<b>CLK_SYS_IO</b>	读写	0x1
12	<b>CLK_SYS_I2C1</b>	读写	0x1
11	<b>CLK_SYS_I2C0</b>	读写	0x1
10	<b>CLK_SYS_HSTX</b>	读写	0x1
9	<b>CLK_HSTX</b>	读写	0x1
8	<b>CLK_SYS_GLITCH_DETECTOR</b>	读写	0x1
7	<b>CLK_SYS_DMA</b>	读写	0x1
6	<b>CLK_SYS_BUSFABRIC</b>	读写	0x1
5	<b>CLK_SYS_BUSCTRL</b>	读写	0x1
4	<b>CLK_SYS_BOOTRAM</b>	读写	0x1
3	<b>CLK_SYS_ADC</b>	读写	0x1
2	<b>CLK_ADC_ADC</b>	读写	0x1
1	<b>CLK_SYS_ACCESSCTRL</b>	读写	0x1
0	<b>CLK_SYS_CLOCKS</b>	读写	0x1

## CLOCKS: SLEEP\_EN1 寄存器

偏移: 0xb8

**描述**

在睡眠模式下启用时钟

表 590. SLEEP\_EN1  
寄存器

位	描述	类型	复位
31	保留。	-	-
30	<b>CLK_SYS_XOSC</b>	读写	0x1
29	<b>CLK_SYS_XIP</b>	读写	0x1
28	<b>CLK_SYS_WATCHDOG</b>	读写	0x1
27	<b>CLK_USB</b>	读写	0x1
26	<b>CLK_SYS_USBCTRL</b>	读写	0x1
25	<b>CLK_SYS_UART1</b>	读写	0x1
24	<b>CLK_PERI_UART1</b>	读写	0x1
23	<b>CLK_SYS_UART0</b>	读写	0x1
22	<b>CLK_PERI_UART0</b>	读写	0x1
21	<b>CLK_SYS_TRNG</b>	读写	0x1
20	<b>CLK_SYS_TIMER1</b>	读写	0x1
19	<b>CLK_SYS_TIMER0</b>	读写	0x1
18	<b>CLK_SYS_TICKS</b>	读写	0x1
17	<b>CLK_REF_TICKS</b>	读写	0x1
16	<b>CLK_SYS_TBMAN</b>	读写	0x1
15	<b>CLK_SYS_SYSINFO</b>	读写	0x1
14	<b>CLK_SYS_SYSCFG</b>	读写	0x1
13	<b>CLK_SYS_SRAM9</b>	读写	0x1
12	<b>CLK_SYS_SRAM8</b>	读写	0x1
11	<b>CLK_SYS_SRAM7</b>	读写	0x1
10	<b>CLK_SYS_SRAM6</b>	读写	0x1
9	<b>CLK_SYS_SRAM5</b>	读写	0x1
8	<b>CLK_SYS_SRAM4</b>	读写	0x1
7	<b>CLK_SYS_SRAM3</b>	读写	0x1
6	<b>CLK_SYS_SRAM2</b>	读写	0x1
5	<b>CLK_SYS_SRAM1</b>	读写	0x1
4	<b>CLK_SYS_SRAM0</b>	读写	0x1
3	<b>CLK_SYS_SPI1</b>	读写	0x1
2	<b>CLK_PERI_SPI1</b>	读写	0x1
1	<b>CLK_SYS_SPI0</b>	读写	0x1
0	<b>CLK_PERI_SPI0</b>	读写	0x1

**CLOCKS: ENABLED0 寄存器**

偏移: 0xbc

**描述**

指示时钟启用状态

表 591. ENABLED0  
寄存器

位	描述	类型	复位
31	<b>CLK_SYS_SIO</b>	只读	0x0
30	<b>CLK_SYS_SHA256</b>	只读	0x0
29	<b>CLK_SYS_PSM</b>	只读	0x0
28	<b>CLK_SYS_ROSC</b>	只读	0x0
27	<b>CLK_SYS_ROM</b>	只读	0x0
26	<b>CLK_SYS_RESETS</b>	只读	0x0
25	<b>CLK_SYS_PWM</b>	只读	0x0
24	<b>CLK_SYS_POWMAN</b>	只读	0x0
23	<b>CLK_REF_POWMAN</b>	只读	0x0
22	<b>CLK_SYS_PLL_USB</b>	只读	0x0
21	<b>CLK_SYS_PLL_SYS</b>	只读	0x0
20	<b>CLK_SYS_PIO2</b>	只读	0x0
19	<b>CLK_SYS_PIO1</b>	只读	0x0
18	<b>CLK_SYS_PIO0</b>	只读	0x0
17	<b>CLK_SYS_PADS</b>	只读	0x0
16	<b>CLK_SYS OTP</b>	只读	0x0
15	<b>CLK_REF OTP</b>	只读	0x0
14	<b>CLK_SYS_JTAG</b>	只读	0x0
13	<b>CLK_SYS IO</b>	只读	0x0
12	<b>CLK_SYS_I2C1</b>	只读	0x0
11	<b>CLK_SYS_I2C0</b>	只读	0x0
10	<b>CLK_SYS_HSTX</b>	只读	0x0
9	<b>CLK_HSTX</b>	只读	0x0
8	<b>CLK_SYS_GLITCH_DETECTOR</b>	只读	0x0
7	<b>CLK_SYS_DMA</b>	只读	0x0
6	<b>CLK_SYS_BUSFABRIC</b>	只读	0x0
5	<b>CLK_SYS_BUSCTRL</b>	只读	0x0
4	<b>CLK_SYS_BOOTRAM</b>	只读	0x0
3	<b>CLK_SYS_ADC</b>	只读	0x0
2	<b>CLK_ADC_ADC</b>	只读	0x0
1	<b>CLK_SYS_ACCESSCTRL</b>	只读	0x0
0	<b>CLK_SYS_CLOCKS</b>	只读	0x0

**CLOCKS: ENABLED1 寄存器**

偏移: 0xc0

**描述**

指示时钟启用状态

表 592. ENABLED1  
寄存器

位	描述	类型	复位
31	保留。	-	-
30	<b>CLK_SYS_XOSC</b>	只读	0x0
29	<b>CLK_SYS_XIP</b>	只读	0x0
28	<b>CLK_SYS_WATCHDOG</b>	只读	0x0
27	<b>CLK_USB</b>	只读	0x0
26	<b>CLK_SYS_USBCTRL</b>	只读	0x0
25	<b>CLK_SYS_UART1</b>	只读	0x0
24	<b>CLK_PERI_UART1</b>	只读	0x0
23	<b>CLK_SYS_UART0</b>	只读	0x0
22	<b>CLK_PERI_UART0</b>	只读	0x0
21	<b>CLK_SYS_TRNG</b>	只读	0x0
20	<b>CLK_SYS_TIMER1</b>	只读	0x0
19	<b>CLK_SYS_TIMER0</b>	只读	0x0
18	<b>CLK_SYS_TICKS</b>	只读	0x0
17	<b>CLK_REF_TICKS</b>	只读	0x0
16	<b>CLK_SYS_TBMAN</b>	只读	0x0
15	<b>CLK_SYS_SYSINFO</b>	只读	0x0
14	<b>CLK_SYS_SYSCFG</b>	只读	0x0
13	<b>CLK_SYS_SRAM9</b>	只读	0x0
12	<b>CLK_SYS_SRAM8</b>	只读	0x0
11	<b>CLK_SYS_SRAM7</b>	只读	0x0
10	<b>CLK_SYS_SRAM6</b>	只读	0x0
9	<b>CLK_SYS_SRAM5</b>	只读	0x0
8	<b>CLK_SYS_SRAM4</b>	只读	0x0
7	<b>CLK_SYS_SRAM3</b>	只读	0x0
6	<b>CLK_SYS_SRAM2</b>	只读	0x0
5	<b>CLK_SYS_SRAM1</b>	只读	0x0
4	<b>CLK_SYS_SRAM0</b>	只读	0x0
3	<b>CLK_SYS_SPI1</b>	只读	0x0
2	<b>CLK_PERI_SPI1</b>	只读	0x0
1	<b>CLK_SYS_SPI0</b>	只读	0x0
0	<b>CLK_PERI_SPI0</b>	只读	0x0

**CLOCKS: INTR 寄存器**

偏移: 0xc4

**说明**

原始中断

表 593. INTR 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLK_SYS_RESUS</b>	只读	0x0

**CLOCKS: INTS 寄存器**

偏移量：0xc8

**说明**

中断使能

表 594. INTS 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLK_SYS_RESUS</b>	读写	0x0

**CLOCKS: INTF 寄存器**

偏移量：0xcc

**说明**

中断强制

表 595. INTF 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLK_SYS_RESUS</b>	读写	0x0

**CLOCKS: INTS 寄存器**

偏移量：0xd0

**说明**

掩码与强制后的中断状态

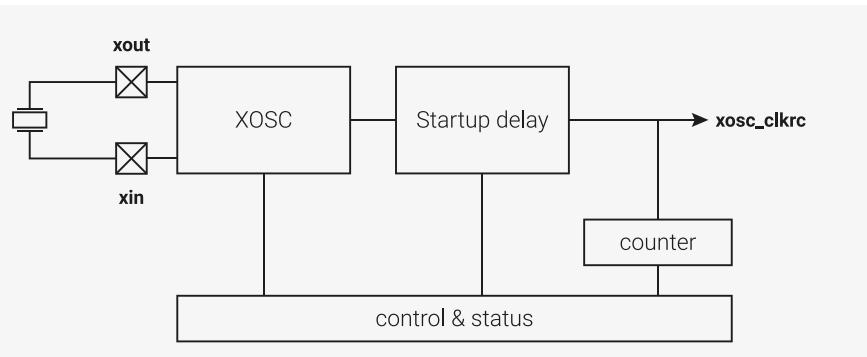
表 596. INTS 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLK_SYS_RESUS</b>	只读	0x0

**8.2. 晶体振荡器 (XOSC)****8.2.1. 概述**

图 38. XOSC 是一款放大器。当压电晶体连接于 XIN 与 XOUT 之间时，放大反馈驱动晶体进入机械谐振状态。由此为片上时钟生成提供了精确的参考。

外部信号亦可直接驱动至 XIN。



晶体振荡器（XOSC）采用外部晶体产生精确参考时钟。RP2350 支持 1 MHz 至 50 MHz 的晶体，RP2350 参考设计（见《RP2350 硬件设计，最小设计示例》）采用 12 MHz 晶体。参考时钟分配至 PLL，PLL 可对 XOSC 频率进行倍频，进而提供精确的高速时钟。例如，它们可以产生一个48 MHz的时钟，满足USB接口对频率精度的要求，同时产生一个最高150 MHz的系统时钟。XOSC时钟亦为时钟发生器的时钟源，如有需要可直接使用。

若用户已有精确时钟源，可直接将外部时钟送入 XIN（又称 XI），并禁用振荡器电路。在此模式下，XIN 输入时钟频率最高可达 50 MHz。

如需外部使用 XOSC 时钟，可通过 `clk_gpclk0` 至 `clk_gpclk3` 发生器，将其输出至 GPIO 引脚。不可直接从 XIN (XI) 或 XOUT (XO) 引脚获取 XOSC 输出。

#### 注意

PLL 最低晶体振荡频率为 5 MHz。详见第 8.6 节，“PLL”。

### 8.2.1.1 推荐晶体振荡器

为确保最佳性能及典型工作温度范围内的稳定性，建议选用 Abracon ABM8-272-T3 晶体振荡器。您可直接向 Abracon 或其授权经销商采购该型号。Abracon ABM8-272-T3 的规格如下：

表 597 主晶体规格

参数	最小值	典型值	最大值	单位	备注
中心频率	12.000	12.000	12.000	MHz	
工作模式	基频-AT	基频-AT	基频-AT		
工作温度	-40		+85	°C	
存储温度	-55		+125	°C	
频率公差 (25 °C)	-30		+30	ppm	
频率稳定性 (25 °C)	-30		+30	ppm	
等效串联电阻 (R1)			50	Ω	
并联电容 (C0)			3.0	pF	
负载电容 (CL)	10	10	10	pF	
驱动等级		10	200	μW	
老化	-5		+5	ppm	@25±3 °C, 第一年
绝缘电阻	500			MΩ	@100 Vdc±15 V

即使使用规格相近的晶体，仍需对电路在不同温度范围内进行测试，

以确保稳定性。

晶体振荡器由VDDIO电压供电。因此，Abracon 晶体及其特定的阻尼电阻均已针对3.3V操作进行调校。若使用不同的 IO 电压，则需重新调校。

任何对晶体参数的更改均可能导致连接晶体电路的组件发生不稳定。

若无法直接从 Abracon 或其经销商处获取推荐晶体，请联系 [applications@raspberrypi.com](mailto:applications@raspberrypi.com)。

Raspberry Pi Pico 2 已专门针对 Abracon ABM8-272-T3 晶体规格调校。有关在 RP2350 上使用晶体的示例，请参阅 Raspberry Pi Pico 2 数据手册附录 B 中的电路原理图及 Raspberry Pi Pico 2 设计文件。

### 8.2.2. 相较于 RP2040 的变更

- 当在CTRL.FREQ\_RANGE中选择适当范围时，最大晶体振荡频率由15 MHz提升至50 MHz。  
[CTRL.FREQ\\_RANGE](#)

#### **i 注意**

上述更改适用于将XOSC作为晶体振荡器时，晶体连接于 **XIN** 和 **XOUT** 引脚之间。当使用XOSC的XIN引脚作为外部振荡器的CMOS时钟输入时，最大频率始终为50 MHz。对于CMOS输入情况，无需配置CTRL.FREQ\_RANGE。CMOS输入行为与RP2040一致。

#### **i 注意**

最大 **clk\_ref** 频率为25 MHz。如果使用频率高于25 MHz的晶体作为 **clk\_ref** 的源，必须通过 **clk\_ref** 分频器分频XOSC输出。

### 8.2.3. 使用方法

XOSC在芯片启动时被禁用，RP2350引导时使用环形振荡器（ROSC）。要启动XOSC，程序员必须设置[CTRL\\_ENABLE](#)寄存器。XOSC不可立即使用，因为振荡需一定时间才能达到足够幅度。该时间依所选晶体而定，通常为数毫秒量级。XOSC包含由[STARTUP\\_DELAY](#)寄存器控制的定时器以自动管理该过程，当XOSC时钟可用时，定时器将设置一个标志（[STATUS\\_STABLE](#)）。

### 8.2.4. 启动延迟

STARTUP\_DELAY寄存器规定必须从晶体观察多少个时钟周期后方可使用。该值以256的倍数表示。SDK中的xosc\_init函数设置该值。对于以12 MHz运行XOSC的RP2350参考设计（参见RP2350硬件设计，最小设计示例），默认的1毫秒延时足够。当定时器到期时，STATUS\_STABLE标志将被设置，表明XOSC输出可用。

启动XOSC之前，程序员必须确保[STARTUP\\_DELAY](#)寄存器已正确配置。所需值可通过以下公式计算：

$$(f_{Crystal} \times t_{Stable}) \div 256$$

因此，对于12 MHz晶体和1毫秒等待时间，计算如下：

$$(12 \times 10^6 \cdot 1 \times 10^{-3}) \div 256 \approx 47$$

**i 注意**

该值向上取整至最接近的整数，因此等待时间将略超过1毫秒。

### 8.2.5. XOSC 计数器

COUNT寄存器提供了一种管理短时软件延迟的方法。使用该方法步骤如下：

1. 向COUNT寄存器写入一个值。寄存器自动以XOSC频率开始倒计数至零。
2. 轮询该寄存器，直至其值为零。

这比在软件循环中使用NOP指令更为优选，因为它不依赖核心时钟频率、编译器以及已编译代码的执行时间。

### 8.2.6. DORMANT 模式

在休眠模式（参见第6.5.3节，“休眠状态”）中，所有片上时钟均可暂停以节省功耗。这对于电池供电的应用尤为有用。RP2350通过中断从休眠模式唤醒：中断源可以是外部事件，如GPIO引脚的边沿触发，或AON定时器触发。此项必须在进入休眠模式前进行配置。要使用AON定时器触发从休眠模式的唤醒，时钟必须来自LPOSC或外部时钟源。

进入休眠（DORMANT）模式：

1. 将所有内部时钟切换为由XOSC或ROSC驱动，并停止锁相环（PLLs）。
2. 选择一个振荡器（XOSC或ROSC），向所选振荡器的 DORMANT 寄存器写入特定32位数值以停止其运行。

退出休眠模式时，所选振荡器将重新启动。若选择XOSC，频率更为精确，但由于启动延迟（RP2350参考设计中>1毫秒，详见《RP2350硬件设计，最简设计示例》），重启时间较长。若选择ROSC，频率精度较低，但启动时间极短（约1μs）。有关导致系统退出休眠模式的事件，请参见章节6.5.3.1“从休眠（DORMANT）状态唤醒”。

**i 注意**

进入休眠模式前，必须停止锁相环（PLLs）。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_xosc/xosc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_xosc/xosc.c) 第56至63行

```

56 void xosc_dormant(void) {
57 // 警告：此操作将停止xosc，直到由中断唤醒
58 xosc_hw->dormant = XOSC_DORMANT_VALUE_DORMANT;
59 // 唤醒后，等待其稳定
60 while(!(xosc_hw->status & XOSC_STATUS_STABLE_BITS)) {
61 tight_loop_contents();
62 }
63 }
```

### ● 警告

如果未在进入DORMANT模式前配置IRQ，则振荡器不会重新启动。

请参见第6.5.6.2节“DORMANT”，其中提供了使用XOSC的DORMANT模式完整示例。

## 8.2.7. 程序员模型

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware\\_structs/include/hardware/structs/xosc.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware_structs/include/hardware/structs/xosc.h) 第27至57行

```

27 typedef struct {
28 _REG_(XOSC_CTRL_OFFSET) // XOSC_CTRL
29 // 晶体振荡器控制
30 // 0x00ffff000 [23:12] ENABLE (-) 上电时该字段初始化为DISABLE，且
31 // 该...
31 // 0x00000fff [11:0] FREQ_RANGE (-) 该12位代码旨在提供一定程度的保护...
32
32 io_rw_32 ctrl;
33
34 _REG_(XOSC_STATUS_OFFSET) // XOSC_STATUS
35 // 晶体振荡器状态
36 // 0x80000000 [31] STABLE (0) 振荡器正在运行且处于稳定状态
37 // 0x01000000 [24] BADWRITE (0) 已向CTRL_ENABLE或...
38
38 // 0x00001000 [12] ENABLED (-) 振荡器已启用，但不一定正在运行
39 // 且...
39 // 0x00000003 [1:0] FREQ_RANGE (-) 当前频率范围设置
40 io_rw_32 status;
41
42 _REG_(XOSC_DORMANT_OFFSET) // XOSC_DORMANT
43 // 晶体振荡器暂停控制
44 // 0xffffffff [31:0] DORMANT (-) 用于通过暂停XOSC以节能 +
45 io_rw_32 dormant;
46
47 _REG_(XOSC_STARTUP_OFFSET) // XOSC_STARTUP
48 // 控制启动延迟
49 // 0x00100000 [20] X4 (-) 启动延迟乘4，以防万一
50 // 0x00003fff [13:0] DELAY (-) 按256*xtal_period的倍数
51 io_rw_32 startup;
52
53 _REG_(XOSC_COUNT_OFFSET) // XOSC_COUNT
54 // 以XOSC频率运行的递减计数器，计数至零后停止。
55 // 0x0000ffff [15:0] COUNT (0x0000)
56 io_rw_32 count;
57 } xosc_hw_t;
```

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_xosc/xosc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_xosc/xosc.c) 第29至43行

```

29 void xosc_init(void) {
30 // 假设输入频率为1-15 MHz，已在上方验证。
31 xosc_hw->ctrl = XOSC_CTRL_FREQ_RANGE_VALUE_1_15MHZ;
32
33 // 设置xosc启动延迟
34 xosc_hw->startup = STARTUP_DELAY;
35
36 // 由于已设置频率范围和启动延迟，现设置使能位
37 hw_set_bits(&xosc_hw->ctrl, XOSC_CTRL_ENABLE_VALUE_ENABLE << XOSC_CTRL_ENABLE_LSB);
38
```

```

39 // 等待XOSC稳定
40 while(!(xosc_hw->status & XOSC_STATUS_STABLE_BITS)) {
41 tight_loop_contents();
42 }
43 }

```

## 8.2.8. 寄存器列表

XOSC寄存器基址为 **0x40048000**（在SDK中定义为XOSC\_BASE）。

表598.  
XOSC寄存器列表

偏移量	名称	说明
0x00	<a href="#">CTRL</a>	晶体振荡器控制
0x04	<a href="#">状态</a>	晶体振荡器状态
0x08	<a href="#">休眠</a>	晶体振荡器暂停控制
0x0c	<a href="#">启动延迟</a>	控制启动延迟
0x10	<a href="#">计数</a>	一个以XOSC频率运行的递减计数器，计数至零后停止。

### XOSC: CTRL寄存器

偏移: 0x00

#### 描述

晶体振荡器控制

表599. CTRL  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:12	<b>ENABLE:</b> 上电时该字段初始化为DISABLE，芯片从ROSC运行。 如果芯片随后被配置为从XOSC运行，则将该字段设置为DISABLE可能导致芯片锁死。若有此顾虑，请从ROSC运行clk_ref并启用clk_sys的RESUS功能。 该12位代码旨在防止意外写入。 无效设置将保留之前的值。实际使用的值可从STATUS_ENABLED读取。	读写	-
	枚举值：		
	0xd1e → DISABLE		
	0xfb → ENABLE		
11:0	<b>FREQ_RANGE:</b> 该12位代码旨在防止意外写入。无效设置将保留之前的值。 实际使用的值可从STATUS_FREQ_RANGE读取。	读写	-
	枚举值：		
	0xaa0 → 1_15MHZ		
	0xaa1 → 10_30MHZ		
	0xaa2 → 25_60MHZ		
	0xaa3 → 40_100MHZ		

## XOSC：状态寄存器

偏移: 0x04

### 描述

晶体振荡器状态

表600. STATUS寄存器

位	描述	类型	复位
31	稳定：振荡器正在运行且处于稳定状态	只读	0x0
30:25	保留。	-	-
24	错误写入：向 CTRL_ENABLE、CTRL_FREQ_RANGE 或 DORMANT 寄存器写入了无效值	WC	0x0
23:13	保留。	-	-
12	启用：振荡器已启用，但不一定正在运行且稳定，复位时默认为0	只读	-
11:2	保留。	-	-
1:0	频率范围：当前的频率范围设置	只读	-
	枚举值：		
	0x0 → 1_15MHZ		
	0x1 → 10_30MHZ		
	0x2 → 25_60MHZ		
	0x3 → 40_100MHZ		

## XOSC：休眠寄存器

偏移: 0x08

### 说明

晶体振荡器暂停控制

表601. DORMANT寄存器

位	描述	类型	复位
31:0	用于通过暂停XOSC以节省功耗 上电时此字段初始化为WAKE 无效写入同样会选择WAKE 警告：选择休眠模式前请先停止PLL 警告：在选择休眠模式之前，请先设置 irq。	读写	-
	枚举值：		
	0x636f6d61 → DORMANT		
	0x77616b65 → WAKE		

## XOSC：STARTUP 寄存器

偏移: 0x0c

### 描述

控制启动延迟

表602. STARTUP寄存器

位	描述	类型	复位
31:21	保留。	-	-

位	描述	类型	复位
20	<b>X4</b> : 将 startup_delay 乘以 4, 以防万一。复位值由可掩码编程的锁存单元控制, 备用于从 XOSC 启动且默认启动延迟不足的情况。	读写	0x0
19:14	保留。	-	-
13:0	<b>DELAY</b> : 以 $256 \times \text{xtal\_period}$ 的倍数计。复位值 0xc4 约等于 50,000 个时钟周期。	读写	0x00c4

## XOSC: COUNT 寄存器

偏移: 0x10

表603. COUNT 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p>一个以 xosc 频率运行的递减计数器, 计数到零后停止。 可用于设置时间敏感硬件时实现短暂的软件延迟。</p> <p>要启动计数器, 请写入非零值。计数器运行时读取返回 1, 计数结束后返回 0。 最小计数值为4。计数值小于4的, 将视为计数值等于4。 请注意, 同步至寄存器时钟域需耗费2个寄存器时钟周期, 计数器无法对此进行补偿。</p>	读写	0x0000

## 8.3. 环形振荡器 (ROSC)

### 8.3.1. 概述

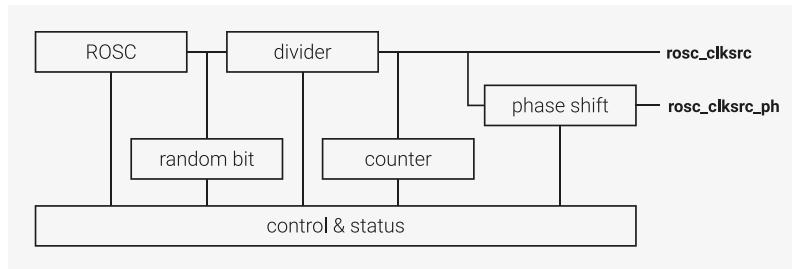
环形振荡器 (ROSC) 是一种由反相器环构成的片上振荡器。该振荡器无需外部组件, 在RP2350上电时自动启动。启动过程中为处理器内核提供时钟。ROSC的频率可编程, 能够直接为内核提供高速时钟, 但其频率随工艺(Process)、电压(Voltage)及温度(Temperature) (PVT) 变化, 因而无法为需高频率精度的组件如AON定时器、USB及ADC提供时钟。频率可随机化, 以防止通过电源轨迹恢复系统时钟的尝试。第8.1节“概述”中讨论了缓解频率变化的方法, 但这些方法仅适用于极低功耗设计。对于大多数需要精确时钟频率的应用, 请切换至XOSC和PLL。启动过程中, ROSC以名义11MHz运行, 并保证在无随机化时频率范围为4.6MHz至19.6MHz, 随机化时范围为4.6MHz至24.0MHz。

#### ① 注意

RP2350 A3及后续版本默认启用随机化, 启动只读存储器 (bootrom) 通过将 DIV降低至2, 使ROSC基频提升四倍。因此, `clk_sys`保证在18.4 MHz至96.0 MHz范围内, `clk_ref`则通过增加除数保持在名义11 MHz。该调整提升了毛刺检测器的灵敏度, 灵敏度与时钟周期成反比, 进而更有效地保护只读存储器的早期启动路径。

芯片启动后, 程序员可选择继续使用ROSC并提升其频率, 或启动晶体振荡器 (XOSC) 和PLL。系统时钟切换至XOSC后, 可禁用ROSC。每种振荡器均具优势; 在它们之间切换, 以实现您的应用的最佳解决方案。

图39. ROSC  
概述。



### 8.3.2. 相较于 RP2040 的变更

新增频率随机化功能。

### 8.3.3. RP2350 修订版本之间的变更

RP2350 A3将FREQA.DS0\_RANDOM和FREQA.DS1\_RANDOM的复位值由 0 改为 1。有关复位时钟配置相关更改的信息，请参见硬件更改。有关A3版Boot ROM中相关更改的信息，请参见Bootrom更改一节。

### 8.3.4. ROSC 与 XOSC 的权衡

ROSC具有以下几个优势：

- 频率可编程，灵活性强
- 低功耗需求
- 无需内部或外部组件
- 可选的频率随机化功能提升安全性

由于ROSC具备可编程频率，能够在不启用PLL的情况下提供快速的核心时钟，并通过时钟发生器分频（第 8.1节，“概述”）产生较慢的外设时钟。ROSC能够立即启动并即时响应频率控制。在进入及退出DORMANT状态时，保持频率设置（参见第6.5.3节，“DORMANT状态”）。但用户必须注意，因电源电压和芯片温度变化，退出DORMANT状态时频率可能发生漂移。

ROSC 的缺点在于其频率随 PVT（工艺、电压和温度）变化，因而不适用于生成精确时钟或软件执行时序关键的应用。然而，PVT 频率变化可被利用，实现自动频率调节以最大化性能。相关内容详见第8.1节，“概述”。

XOSC 唯一的优点是频率精确，但在许多应用中这是不可替代的核心需求。

XOSC 存在以下缺点：

- 需要外部元件（如晶体等）
- 功耗较高
- 启动时间较长 (>1毫秒)
- 频率固定且较低

必须通过 PLL 产生更高频率时钟。PLL 消耗更多功率且启动或频率切换所需时间较长。退出休眠（DORMANT）模式远慢于ROSC，因需重新启动 XOSC 并重新配置 PLL。

### 8.3.5. 频率调整

ROSC 设计为8级，每级均具可编程驱动能力。ROSC 提供两种控制频率的方法。频率范围控制 ROSC 环路中的级数，**FREQ\_A**和**FREQ\_B**寄存器控制各级的驱动强度。

要更改频率范围，请写入 **FREQ\_RANGE**寄存器，该寄存器控制 ROSC 环路中的级数。

**FREQ\_RANGE**寄存器支持以下配置：

表 604. ROSC  
级数范围

名称	级数数量	范围（级数）
低	8	0-7
中	6	2-7
高	4	4-7
超高	2	6-7

逐步更改 **FREQ\_RANGE**寄存器，直至达到所需范围。增加频率范围时，ROSC 输出不会产生抖动，输出时钟可继续使用。降低频率范围时，ROSC 输出会产生抖动，必须为由 ROSC 作为时钟源的模块选择备用时钟源，或在转换期间将其复位。

该行为尚未完全表征，但中范围约为低范围的 1.33 倍，高范围为低范围的 2 倍，超高范围为低范围的 4 倍。**TOOHIGH** 范围名称准确。不应使用该频率，因为 ROSC 的内部逻辑无法在该频率下运行。

**FREQ\_A** 和 **FREQ\_B** 寄存器控制 ROSC 环路中各级的驱动强度。随着驱动强度增加，级内延迟减小，振荡频率提高。每一级设有 3 个驱动强度控制位。每个位开启一个额外驱动，因此每一级具有 4 种驱动强度设置，等同于已设置位数，0 为默认，1 为双驱动，2 为三驱动，3 为四驱动。额外驱动对频率影响呈非线性：第二个驱动效果低于第一个，第三个驱动效果低于第二个，依此类推。为确保切换平滑，每次请仅调整一个驱动强度位。当 **FREQ\_RANGE** 缩短 ROSC 环路时，被绕过的级仍然传递信号，因此其驱动强度必须至少保持不低于环路内最低驱动强度。此操作不会影响振荡频率。

### 8.3.6. 频率随机化

通过将ROSC环路前两个阶段的驱动强度控制设置为 **DS0\_RANDOM**和 **DS1\_RANDOM**，即可启用随机化。LFSR随后为这两个阶段提供驱动强度控制，无论**FREQ\_RANGE**如何设置，这两个阶段始终包含在环路中。建议对两个阶段均启用随机化。当选择低**FREQ\_RANGE**时，随机化器将使频率增加高达默认值的22%。如果仅随机化其中一个阶段，频率提升约为该值的一半。LFSR可通过写入**RANDOM**寄存器进行种子初始化。此操作可随时执行，但会重启随机化器。

### 8.3.7. ROSC 分频器

ROSC频率过快，无法直接使用，因此通过由 **DIV**寄存器控制的整数分频器进行分频。您可在ROSC运行时更改 **DIV**，输出时钟频率将无毛刺地变化。默认除数为8，以确保芯片启动时输出时钟处于指定范围内。

分频器具有两个输出，分别为**rosc\_clksrc**和**rosc\_clksrc\_ph**。**rosc\_clksrc\_ph**是**rosc\_clksrc**的相位偏移版本。此功能主要用于产品开发阶段；若 **PHASE**寄存器保持默认状态，两个输出将完全相同。

### 8.3.8. 随机数发生器

当系统时钟由XOSC驱动时，可使用ROSC生成随机数。启用ROSC并读取 **RANDOMBIT** 寄存器以获取单比特随机数；若需获取  $n$  比特随机值，则需读取该寄存器  $n$  次。此方法不满足安全系统对随机性的要求，因其易受攻击，但在较低风险应用中仍可能适用。若核心由ROSC驱动，则读取值不具随机性，因为寄存器读取时间与ROSC相位相关。

### 8.3.9. ROSC 计数器

**COUNT** 寄存器提供了一种管理短时软件延迟的方法。使用该方法步骤如下：

1. 向 **COUNT** 寄存器写入一个值。该寄存器自动开始以ROSC频率倒计时至零。
2. 轮询该寄存器，直至其值为零。

这比在软件循环中使用NOP指令更为优选，因为它不依赖核心时钟频率、编译器以及已编译代码的执行时间。

### 8.3.10. DORMANT 模式

在休眠模式（参见第6.5.3节，“休眠状态”）中，所有片上时钟均可暂停以节省功耗。这对于电池供电的应用尤为有用。RP2350通过中断从休眠模式唤醒：中断源可以是外部事件，如GPIO引脚的边沿触发，或AON定时器触发。此项必须在进入休眠模式前进行配置。要使用AON定时器触发从休眠模式的唤醒，时钟必须来自LPOSC或外部时钟源。

进入休眠（DORMANT）模式：

1. 将所有内部时钟切换为由XOSC或ROSC驱动，并停止锁相环（PLLs）。
2. 选择一个振荡器（XOSC或ROSC），向所选振荡器的 **DORMANT** 寄存器写入特定32位数值以停止其运行。

退出休眠模式时，所选振荡器将重新启动。如果选择XOSC，频率将更为精确，但因启动延迟，重启时间较长（RP2350参考设计中启动延迟超过1毫秒，详见RP2350硬件设计——最小设计示例）。如果选择ROSC，频率较不精确，但启动时间极短（约1微秒）。有关导致系统退出休眠模式的事件，请参见章节6.5.3.1“从休眠（DORMANT）状态唤醒”。

#### 注意

进入休眠模式前，必须停止锁相环（PLLs）。

Pico Extras: [https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\\_common/hardware\\_rosc/rosc.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/hardware_rosc/rosc.c) 第56至61行

```

56 void rosc_set_dormant(void) {
57 // 警告：此操作将停止rosc，直至通过irq唤醒
58 rosc_write(&rosc_hw->dormant, ROSC_DORMANT_VALUE_DORMANT);
59 // 唤醒后等待osc稳定
60 while(!(rosc_hw->status & ROSC_STATUS_STABLE_BITS));
61 }
```

**● 警告**

如果未在进入DORMANT模式前配置IRQ，则振荡器不会重新启动。

请参见第6.5.6.2节，“DORMANT”，其中包含休眠模式的一些示例。

### 8.3.11. 寄存器列表

ROSC寄存器的基地址为 `0x400e8000`（在SDK中定义为ROSC\_BASE）。

表605.  
ROSC寄存器列表

偏移量	名称	说明
0x00	<a href="#">CTRL</a>	环形振荡器控制
0x04	<a href="#">FREQA</a>	环形振荡器频率控制A
0x08	<a href="#">FREQB</a>	环形振荡器频率控制B
0x0c	<a href="#">RANDOM</a>	向LFSR随机数发生器加载一个值
0x10	<a href="#">休眠</a>	环形振荡器暂停控制
0x14	<a href="#">DIV</a>	控制输出分频器
0x18	<a href="#">PHASE</a>	控制相位偏移输出
0x1c	<a href="#">状态</a>	环形振荡器状态
0x20	<a href="#">RANDOMBIT</a>	返回1位随机值
0x24	<a href="#">计数</a>	一个以ROSC频率运行的向下计数器，计数至零后停止。

### ROSC: CTRL寄存器

偏移: 0x00

#### 描述

环形振荡器控制

表606. CTRL  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:12	<b>ENABLE:</b> 此字段在上电时初始化为ENABLE 必须在将此字段设置为DISABLE之前切换系统时钟至其他来源，否则芯片将锁死 该12位代码旨在防止意外写入。 无效设置将使振荡器启用。	读写	-
	枚举值：		
	0xd1e → DISABLE		
	0xfab → ENABLE		

位	描述	类型	复位
11:0	<b>FREQ_RANGE:</b> 控制ROSC环中延迟阶段的数量 LOW使用阶段0至7 MEDIUM使用阶段0至5 HIGH使用阶段0至3 TOOHIGH使用阶段0至1，频率超出设计规范，故不应使用  时钟输出在频率范围逐步提高时不会产生毛刺 时钟输出在频率范围降低时会产生毛刺 注意：此处数值采用格雷码编码，因此HIGH位于TOOHIGH之前	读写	0xaa0
	枚举值：		
	0xfa4 → LOW		
	0xfa5 → 中等		
	0xfa7 → 高		
	0xfa6 → 过高		

## ROSC: FREQA 寄存器

偏移: 0x04

### 描述

FREQA 和 FREQB 寄存器通过控制各阶段的驱动强度来调节频率。驱动强度分为四档，由设置的位数决定。增加设置的位数将提高驱动强度及振荡频率。

- 0 位设置为默认驱动强度
- 1 位设置时驱动强度翻倍
- 2 位设置时驱动强度增至三倍
- 3 位设置时驱动强度增至四倍

频率随机化时，请同时设置 DS0\_RANDOM=1 和 DS1\_RANDOM=1

表 607. FREQA 寄存器

位	描述	类型	复位
31:16	<b>PASSWD:</b> 设置为 0x9696 以应用该设置 该字段的任何其他值将导致所有驱动强度被设为 0	读写	0x0000
	枚举值：		
	0x9696 → 通过		
15	保留。	-	-
14:12	<b>DS3:</b> 阶段 3 驱动强度	读写	0x0
11	保留。	-	-
10:8	<b>DS2:</b> 阶段 2 驱动强度	读写	0x0
7	<b>DS1_RANDOM:</b> 随机化阶段1驱动强度	读写	0x1
6:4	<b>DS1:</b> 阶段1驱动强度	读写	0x0
3	<b>DS0_RANDOM:</b> 随机化阶段0驱动强度	读写	0x1
2:0	<b>DS0:</b> 阶段0驱动强度	读写	0x0

## ROSC: FREQB 寄存器

偏移: 0x08

**描述**

详细说明请参见 freqa 寄存器

表608. FREQB  
寄存器

位	描述	类型	复位
31:16	<b>PASSWD</b> : 设置为 0x9696 以应用该设置 该字段的任何其他值将导致所有驱动强度被设为 0	读写	0x0000
	枚举值:		
	0x9696 → 通过		
15	保留。	-	-
14:12	<b>DS7</b> : 阶段7驱动强度	读写	0x0
11	保留。	-	-
10:8	<b>DS6</b> : 阶段6驱动强度	读写	0x0
7	保留。	-	-
6:4	<b>DS5</b> : 阶段5驱动强度	读写	0x0
3	保留。	-	-
2:0	<b>DS4</b> : 阶段4驱动强度	读写	0x0

**ROSC: RANDOM 寄存器**

偏移: 0x0c

**描述**

向LFSR随机数发生器加载一个值

表609. RANDOM  
寄存器

位	描述	类型	复位
31:0	<b>SEED</b>	读写	0x3f04b16d

**ROSC: DORMANT 寄存器**

偏移: 0x10

**描述**

环形振荡器暂停控制

表610. DORMANT  
寄存器

位	描述	类型	复位
31:0	此功能用于通过暂停ROSC以节省电能 上电时此字段初始化为WAKE 无效写入同样会选择WAKE 警告: 请在选择休眠模式之前设置irq	读写	-
	枚举值:		
	0x636f6d61 → DORMANT		
	0x77616b65 → WAKE		

**ROSC: DIV寄存器**

偏移: 0x14

**说明**

控制输出分频器

表611. DIV  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	设置为 $0xaaa00 + div$ , 其中 <div>=0时除以128<div>=1-127时除以div其他任意值时, div设置为128该寄存器复位时div为32</div></div>	读写	-
	枚举值:		
	$0xaaa00 \rightarrow$ 通过		

## ROSC: PHASE 寄存器

偏移量: 0x18

### 说明

控制相位偏移输出

表612. PHASE  
寄存器

位	描述	类型	复位
31:12	保留。	-	-
11:4	<b>PASSWD</b> : 设置为0xaa 其他值将启用输出, 且shift为0	读写	0x00
3	<b>ENABLE</b> : 使能相位偏移输出 此设置可动态修改	读写	0x1
2	<b>FLIP</b> : 反转相位偏移输出 div=1时忽略此设置	读写	0x0
1:0	<b>SHIFT</b> : 将相位偏移输出相位移位SHIFT个输入时钟周期 此设置可动态修改 在设置 div=1 之前, 必须先将其设为 0	读写	0x0

## ROSC: STATUS 寄存器

偏移: 0x1c

### 描述

环形振荡器状态

表 613. STATUS  
寄存器

位	描述	类型	复位
31	稳定: 振荡器正在运行且处于稳定状态	只读	0x0
30:25	保留。	-	-
24	<b>BADWRITE</b> : 向 CTRL_ENABLE、CTRL_FREQ_RANGE、FREQA、FREQB、DIV、PHASE 或 DORMANT 寄存器写入了无效值	WC	0x0
23:17	保留。	-	-
16	<b>DIV_RUNNING</b> : 后分频器正在运行 该值复位为 0, 但在芯片启动期间会变为 1	只读	-
15:13	保留。	-	-
12	<b>ENABLED</b> : 振荡器已启用, 但不一定正在运行且稳定 该值复位为 0, 但在芯片启动期间会变为 1	只读	-
11:0	保留。	-	-

## ROSC: RANDOMBIT 寄存器

偏移: 0x20

表 614.  
RANDOMBIT 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	此寄存器仅读取振荡器输出状态，若环形振荡器停止或以总线频率的谐波运行，随机性将受损	只读	0x1

## ROSC: COUNT 寄存器

偏移: 0x24

表 615. COUNT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	一个以ROSC频率运行的向下计数器，计数至零后停止。  要启动计数器，请写入非零值。 可用于设置时间敏感硬件时实现短暂的软件延迟。	读写	0x0000

## 8.4. 低功耗振荡器 (LPOSC)

低功耗振荡器 (LPOSC) 在主晶体振荡器关闭以进入低功耗 (P1.x) 状态时，为始终打开的逻辑电路提供时钟信号。它以标称32.768kHz频率运行，是一种RC振荡器，无需外部元件。振荡器的输出时钟用于控制芯片初始启动的时序，以及低功耗状态之间的转换。该振荡器亦可供始终开启定时器(AON Timer)使用，详见第12.10节“始终开启定时器”。

振荡器在核心电源供应可用且上电复位解除后即开始启动。若启用欠压检测，检测到核心电源欠压时振荡器将被禁用；但核心电源恢复且欠压复位解除后，振荡器将重新启动。振荡器频率约需1毫秒稳定，在此期间芯片将保持复位状态。

### 8.4.1. 频率精度与校准

低功耗振荡器初始频率精度为±20%。然而，可通过LPOSC寄存器中的TRIM字段将其调节至±1.5%。提供63个调节档位，每档调节范围为振荡器初始频率的1%至3%。频率可降低32步或提高31步。详见表616，“低功耗振荡器输出频率及校准”，以及第8.4.3节“寄存器列表”。

表616。低功耗  
振荡器输出  
频率及  
校准

参数	描述	最小值	典型值	最大值	单位
$F_{0,\text{initial}}$	初始输出频率	26.2144	32.768	39.3216	kHz
校准步进	频率校准步进	-	1	3	占初始输出频率的百分比
$F_{0,\text{trimmed}}$	校准后输出频率	32.27648	32.768	33.25952	kHz

频率随温度漂移：±14%。

频率随电源电压漂移： $\pm 20\%$ 。

### 8.4.2. 使用外部低功率时钟

可不使用低功耗RC振荡器，而改由GPIO 12、14、20或22之一提供外部32.768 kHz低功耗时钟信号。或者，这些GPIO可用于提供1 kHz或1 Hz时钟脉冲。详见第12.10.5.2节“使用外部时钟替代低功耗振荡器”及第12.10.7节“利用GPIO输入外部时钟或时钟脉冲”。

### 8.4.3. 寄存器列表

低功耗振荡器与始终通电域中的其他电源管理子系统共享寄存器地址空间。本文档其他部分将该地址空间称为POWMAN。第6.4节“电源管理（POWMAN）寄存器”中提供了POWMAN寄存器的完整列表，但与低功耗振荡器相关的寄存器信息在此处重复说明。

POWMAN寄存器的基址起始于 `0x40100000`（在SDK中定义为POWMAN\_BASE）。

- [LPOSC](#)
- [EXT\\_TIME\\_REF](#)
- [LPOSC\\_FREQ\\_KHZ\\_INT](#)
- [LPOSC\\_FREQ\\_KHZ\\_FRAC](#)

## 8.5. 计时发生器

### 8.5.1. 概述

时钟发生器为多个模块提供时间参考：

- 系统定时器：TIMER0和TIMER1（第12.8节，“系统定时器”）
- RISC-V平台定时器（第3.1.8节，“RISC-V平台定时器”）
- Arm Cortex-M33核心0和核心1的SysTick定时器
- 看门狗定时器（第12.9节，“看门狗”）

时钟脉冲（**tick**）是一种周期性信号，为定时器或计数器提供时间基准。这些信号类似于时钟，但并不驱动芯片上任何寄存器的时钟输入。使用时钟滴答（**ticks**）而非时钟信号，使得分发独立于任何子系统时钟的时间基准信息更加简便。例如，系统定时器（TIMER0 和 TIMER1）应当即使系统时钟根据处理器需求变化，仍每微秒持续计数一次。

时钟滴答发生器以 `clk_ref` 作为参考时钟（详见第8.1节“概述”，该节包括 `clk_ref` 在内的系统级时钟概述）。理想情况下，应将 `clk_ref` 配置为使用晶体振荡器（第8.2节“晶体振荡器（XOSC）”）以提供精准参考。发生器内部对 `clk_ref` 进行分频，以生成各目标的时钟滴答信号。

SDK期望系统定时器和RISC-V平台定时器具备名义上的1  $\mu$ s时间基准。同样，Cortex-M33 SysTick定时器要求1  $\mu$ s时间基准，以匹配SYST\_CALIB寄存器中硬连线的100,000值，此为标准Arm软件用于校准SysTick延迟的参数。然而，如果您的软件具有特定要求，例如对24位SysTick外设设置更长的最大延迟，则可能需要对这些时间基准进行不同的缩放。计时器生成器可以独立缩放每个目标的计时基准，互不影响。

对于12 MHz参考时钟，将周期计数设置为12以生成1  $\mu$ s的时钟周期。1 MHz时钟的周期为1  $\mu$ s，因此，

硬件需要计数的时钟周期数为12倍，才能从运行于 $12 \times 1\text{ MHz}$ 的参考时钟获得 $1\mu\text{s}$ 的时钟周期。

在更改周期计数之前，必须使用TIMER0\_CTRL.ENABLE位停止计时器生成器。配置完成后，方可重新启用。

### 8.5.2. 寄存器列表

计时器生成器寄存器起始地址为 `0x40108000`（在SDK中定义为TICKS\_BASE）。

表617. TICKS寄存器列表

偏移量	名称	说明
0x00	PROC0_CTRL	计时器生成器控制寄存器
0x04	PROC0_CYCLES	
0x08	PROC0_COUNT	
0x0c	PROC1_CTRL	计时器生成器控制寄存器
0x10	PROC1_CYCLES	
0x14	PROC1_COUNT	
0x18	TIMER0_CTRL	计时器生成器控制寄存器
0x1c	TIMER0_CYCLES	
0x20	TIMER0_COUNT	
0x24	TIMER1_CTRL	计时器生成器控制寄存器
0x28	TIMER1_CYCLES	
0x2c	TIMER1_COUNT	
0x30	WATCHDOG_CTRL	计时器生成器控制寄存器
0x34	WATCHDOG_CYCLES	
0x38	WATCHDOG_COUNT	
0x3c	RISCV_CTRL	计时器生成器控制寄存器
0x40	RISCV_CYCLES	
0x44	RISCV_COUNT	

#### TICKS: PROC0\_CTRL寄存器

偏移: 0x00

##### 说明

计时器生成器控制寄存器

表618。  
PROC0\_CTRL 寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RUNNING:</b> 时钟发生器当前是否运行？	只读	-
0	<b>ENABLE:</b> 启动 / 停止时钟信号生成	读写	0x0

#### TICKS: PROC0\_CYCLES 寄存器

偏移: 0x04

表 619。  
PROC0\_CYCLES  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	下一次时钟信号产生前的总 clk_tick 周期数。	读写	0x000

## TICKS: PROC0\_COUNT 寄存器

偏移: 0x08

表 620。  
PROC0\_COUNT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	倒计时计数器：下一次时钟信号产生前剩余的 clk_tick 周期数。	只读	-

## TICKS: PROC1\_CTRL 寄存器

偏移: 0x0c

### 描述

计时器生成器控制寄存器

表 621。  
PROC1\_CTRL 寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RUNNING:</b> 时钟发生器当前是否运行？	只读	-
0	<b>ENABLE:</b> 启动 / 停止时钟信号生成	读写	0x0

## TICKS: PROC1\_CYCLES 寄存器

偏移: 0x10

表 622。  
PROC1\_CYCLES  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	下一次时钟信号产生前的总 clk_tick 周期数。	读写	0x000

## TICKS: PROC1\_COUNT 寄存器

偏移: 0x14

表 623。  
PROC1\_COUNT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	倒计时计数器：下一次时钟信号产生前剩余的 clk_tick 周期数。	只读	-

## TICKS: TIMER0\_CTRL 寄存器

偏移量: 0x18

### 说明

计时器生成器控制寄存器

表 624。  
TIMER0\_CTRL 寄存器

位	描述	类型	复位
31:2	保留。	-	-

位	描述	类型	复位
1	<b>RUNNING</b> : 时钟发生器当前是否运行?	只读	-
0	<b>ENABLE</b> : 启动 / 停止时钟信号生成	读写	0x0

## TICKS: TIMER0\_CYCLES 寄存器

偏移: 0x1c

表 625。  
TIMER0\_CYCLES  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	下一次时钟信号产生前的总 clk_tick 周期数。	读写	0x000

## TICKS: TIMER0\_COUNT 寄存器

偏移: 0x20

表 626。  
TIMER0\_COUNT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	倒计时计数器: 下一次时钟信号产生前剩余的 clk_tick 周期数。	只读	-

## TICKS: TIMER1\_CTRL 寄存器

偏移量: 0x24

### 说明

计时器生成器控制寄存器

表 627。  
TIMER1\_CTRL 寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RUNNING</b> : 时钟发生器当前是否运行?	只读	-
0	<b>ENABLE</b> : 启动 / 停止时钟信号生成	读写	0x0

## TICKS: TIMER1\_CYCLES 寄存器

偏移量: 0x28

表 628。  
TIMER1\_CYCLES  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	下一次时钟信号产生前的总 clk_tick 周期数。	读写	0x000

## TICKS: TIMER1\_COUNT 寄存器

偏移: 0x2c

表 629。  
TIMER1\_COUNT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	倒计时计数器：下一次时钟信号产生前剩余的 clk_tick 周期数。	只读	-

## TICKS: WATCHDOG\_CTRL 寄存器

偏移量: 0x30

### 描述

计时器生成器控制寄存器

表 630。  
WATCHDOG\_CTRL  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RUNNING</b> : 时钟发生器当前是否运行？	只读	-
0	<b>ENABLE</b> : 启动 / 停止时钟信号生成	读写	0x0

## TICKS: WATCHDOG\_CYCLES 寄存器

偏移量: 0x34

表 631。  
WATCHDOG\_CYCLES  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	下一次时钟信号产生前的总 clk_tick 周期数。	读写	0x000

## TICKS: WATCHDOG\_COUNT 寄存器

偏移量: 0x38

表 632。  
WATCHDOG\_COUNT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	倒计时计数器：下一次时钟信号产生前剩余的 clk_tick 周期数。	只读	-

## TICKS: RISCV\_CTRL 寄存器

偏移: 0x3c

### 描述

计时器生成器控制寄存器

表 633。  
RISCV\_CTRL 寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RUNNING</b> : 时钟发生器当前是否运行？	只读	-
0	<b>ENABLE</b> : 启动 / 停止时钟信号生成	读写	0x0

## TICKS: RISCV\_CYCLES 寄存器

偏移量: 0x40

表 634。  
RISCV\_CYCLES  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	下一次时钟信号产生前的总 clk_tick 周期数。	读写	0x000

## TICKS: RISCV\_COUNT 寄存器

偏移: 0x44

表 635。  
RISCV\_COUNT  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8:0	倒计时计数器：下一次时钟信号产生前剩余的 clk_tick 周期数。	只读	-

## 8.6. PLL

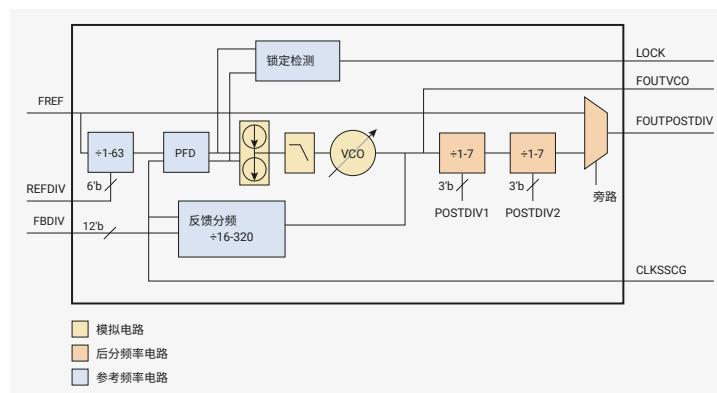
### 8.6.1. 概述

PLL采用参考时钟，并通过带反馈回路的电压控制振荡器（VCO）进行倍频。VCO工作于高频率：750 MHz至1600 MHz之间。因此，有两个后分频器，可在将VCO频率分配至芯片上的时钟发生器之前进行分频。

RP2350中设有两个PLL，它们分别是：

- `pll_sys` - 用于生成最高150 MHz的系统时钟
- `pll_usb` - 用于生成48 MHz的USB参考时钟

图40。两个PLL中，REF（参考）输入连接至晶体振荡器的XIN（XI）输入端。PLL包含一个VCO，通过反馈环路（相位-频率检测器和环路滤波器）锁定参考时钟的固定比率。该结构可合成极高频率，随后可由后置分频器进行分频。



PLL与系统级时钟之间的路由具有灵活性。例如，可通过系统PLL的分频信号驱动USB（例如 $144 \text{ MHz} / 3 = 48 \text{ MHz}$ ），同时释放USB PLL供HSTX外围设备或GPIO上的通用时钟输出等其他用途。

### 8.6.2. 相较于 RP2040 的变化

- RP2350新增当PLL失锁时触发的中断，详见CS.LOCK\_N。

### 8.6.3. PLL 参数计算

要配置PLL，必须确认参考时钟频率，该时钟直接由晶体振荡器引出。这通常是12 MHz的晶体，以确保与RP2350的USB只读存储器（bootrom）兼容。PLL的最终输出频率 $F_{OUTPOSTDIV}$ 可按公式  $(F_{REF} / REFDIV) \times FBDIV / (POSTDIV1 \times POSTDIV2)$  计算。在确定目标输出频率的前提下，须依据PLL设计的以下约束条件选择PLL参数：

- 最小参考频率 ( $F_{REF} / REFDIV$ ) 不得低于5 MHz
- 振荡器频率 ( $F_{OUTVCO}$ ) 应保持在750 MHz至1600 MHz之间
- 反馈分频器 ( $FBDIV$ ) 须介于16至320之间
- 后置分频器  $POSTDIV1$  和  $POSTDIV2$  的取值范围为1至7
- 最大输入频率 ( $F_{REF} / REFDIV$ ) 等于VCO频率除以16，此限制源于反馈分频器的最小值要求

您亦须遵守芯片时钟发生器的最大频率限制（连接至 $F_{OUTPOSTDIV}$ ）。系统PLL的最大频率为150 MHz，USB PLL为48 MHz。若使用频率低于75 MHz的晶体振荡器，且假设VCO频率在1200 MHz至1600 MHz之间，则 $REFDIV$ 应设为1。若使用高速晶体且VCO频率偏低，可能需要增加参考分频器以确保PLL输入频率处于合适范围内。

#### 提示

当  $POSTDIV1$  与  $POSTDIV2$  需赋予不同数值时，应将较大数值指定给  $POSTDIV1$  以降低功耗。

在RP2350参考设计中（见RP2350硬件设计，最简设计示例），将12 MHz晶体连接至晶体振荡器，最小VCO频率为 $12 \text{ MHz} \times 63 = 756 \text{ MHz}$ ，最大VCO频率为 $12 \text{ MHz} \times 133 = 1596 \text{ MHz}$ 。因此， $FBDIV$ 须保持在63到133之间，以免超出VCO频率支持范围。将 $FBDIV$ 设置为100将合成1200 MHz的VCO频率。 $POSTDIV1$ 值为6且 $POSTDIV2$ 值为2将总共除以12，产生PLL最终输出的纯净100 MHz信号。

#### 8.6.3.1. 抖动与功耗

通常，有多组PLL配置参数可实现所需的输出频率（或近似值）。

您需决定是优先降低功耗，还是优先降低抖动——PLL输出时钟周期的逐周期变化。由于可使用更高的后除数值，抖动随VCO频率的升高而减小。请参考以下示例：

- 1500 MHz VCO / 6 / 2 = 125 MHz
- 750 MHz VCO / 6 / 1 = 125 MHz

1500 MHz配置功耗最高，但抖动最小。750 MHz配置功耗最低，但抖动最大。

您可以略微调整所需的输出频率，通过将输出频率调整为输入频率的更接近的有理倍数，从而允许使用更低的VCO频率。某些频率在可能的VCO频率和分频器组合下根本无法实现。

由于RP2350的数字逻辑能够补偿系统时钟上的最严重抖动，因此这不会影响系统稳定性。然而，应用通常需要高精度时钟以满足USB规范中定义的最大允许抖动要求，从而确保数据传输的准确性。

#### 8.6.3.2 使用vcocalc.py计算参数

SDK提供了一个Python脚本，用于搜索所需输出频率的最佳VCO和后级分频选项：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/scripts/vcocalc.py](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/scripts/vcocalc.py)

```

1 #!/usr/bin/env python3
2
3 import argparse
4 import sys
5
6 # 固定硬件参数
7 fbdv_range = range(16, 320 + 1)
8 postdiv_range = range(1, 7 + 1)
9 ref_min = 5
10 refdiv_min = 1
11 refdiv_max = 63
12
13 def validRefdiv(string):
14 if ((int(string) < refdiv_min) or (int(string) > refdiv_max)):
15 raise ValueError("REFDIV 必须在 {} 到 {} 之间".format(refdiv_min, refdiv_max)) 16
16 return int(
17 string)
18
19 parser = argparse.ArgumentParser(description="PLL 参数计算器")
20 parser.add_argument("--input", "-i", default=12, help="输入 (参考) 频率。默认值为 12 MHz", type=float)
21 parser.add_argument("--ref-min", default=5, help="覆盖最小参考频率, 默认值为 5 MHz", type=float)
22 parser.add_argument("--vco-max", default=1600, help="覆盖最大 VCO 频率, 默认值为 1600 MHz", type=float)
23 parser.add_argument("--vco-min", default=750, help="覆盖最低VCO频率, 默认750 MHz", type=float)
24
25 parser.add_argument("--cmake", action="store_true", help="输出可应用所选PLL参数至程序的CMake代码段")
26 parser.add_argument("--cmake-only", action="store_true", help="功能同--cmake, 但仅输出CMake内容, 不打印其他信息")
27 parser.add_argument("--cmake-executable-name", default=<program>, help="设置生成CMake输出中使用的可执行文件名称")
28 parser.add_argument("--lock-refdiv", help="将REFDIV锁定至指定范围内的数值: {} 至 {}".format(refdiv_m
29 in, refdiv_max), type=validRefdiv)
30 parser.add_argument("--low-vco", "-l", action="store_true", help="尽可能使用较低的VCO频率, 以降低功耗, 但会增加抖动")
31 parser.add_argument("output", help="输出频率, 单位MHz", type=float)
32
33 args = parser.parse_args()
34
35 refdiv_range = range(refdiv_min, max(refdiv_min, min(refdiv_max, int(args.input / args
36 .ref_min))) + 1)
37 if args.lock_refdiv:
38 print("锁定 REFDIV 为", args.lock_refdiv)
39 refdiv_range = [args.lock_refdiv]
40
41 best = (0, 0, 0, 0, 0, 0)
42 best_margin = args.output
43
44 for refdiv in refdiv_range:
45 for fbdv in fbdv_range:
46 vco = args.input / refdiv * fbdv
47 if vco < args.vco_min or vco > args.vco_max:
48 continue
49 # pd1 为内层循环, 优先考虑较高的 pd1:pd2 比例
50 for pd2 in postdiv_range:
51 for pd1 in postdiv_range:
52 out = vco / pd1 / pd2
53 margin = abs(out - args.output)
54 vco_is_better = vco < best[5] if args.low_vco else vco > best[5]
55 if ((vco * 1000) % (pd1 * pd2)):
```

```

51 continue
52 if margin < best_margin or (abs(margin - best_margin) < 1e-9 and
53 vco_is_better):
54 best = (out, fbdv, pd1, pd2, refdiv, vco)
55 best_margin = margin
56
57 best_out, best_fbdv, best_pd1, best_pd2, best_refdiv, best_vco = best
58
59 if best[0] > 0:
60 cmake_output = \
61 f"""target_compile_definitions({args.cmake_executable_name} PRIVATE
62 PLL_SYS_REFDIV={best_refdiv}
63 PLL_SYS_VCO_FREQ_HZ={int((args.input * 1_000_000) / best_refdiv * best_fbdv)}
64 PLL_SYS_POSTDIV1={best_pd1}
65 PLL_SYS_POSTDIV2={best_pd2}
66 SYS_CLK_HZ={int((args.input * 1_000_000) / (best_refdiv * best_pd1 * best_pd2) *
67 best_fbdv)}"""
67 """
68 if not args.cmake_only:
69 print("请求频率: {} MHz".format(args.output))
70 print("实际频率: {} MHz".format(best_out))
71 print("REFDIV: {}".format(best_refdiv))
72 print("FBDIV: {} (VCO = {} MHz)".format(best_fbdv, args.input / best_refdiv *
73 best_fbdv))
74 print("PD1: {}".format(best_pd1))
75 print("PD2: {}".format(best_pd2))
76 if best_refdiv != 1:
77 print(
78 "\n这需要一个非默认的REFDIV值。\\n"
79 "请在您的CMakeLists.txt中添加以下内容以应用REFDIV:\\n"
80)
81 elif args.cmake or args.cmake_only:
82 print("")
83 print(cmake_output)
84 else:
85 sys.exit("未找到解决方案")

```

给定输入和输出频率，脚本将寻找最佳的PLL参数组合。当脚本找到多个同等优良的组合时，将返回产生最高VCO频率的参数，以确保最佳输出稳定性。传递 `-l` 或 `--low-vco` 标志以优先选择较低频率，从而降低功耗。传递 `--vco-max` 标志以限制最大VCO频率。若脚本无法依据提供的约束找到精确匹配，则输出最接近的合理匹配。

以下示例演示如何使用脚本请求48 MHz输出以实现最佳输出稳定性：

```

$./vcocalc.py 48
请求频率: 48.0 MHz
实现频率: 48.0 MHz
REFDIV: 1
FBDIV: 120 (VCO = 1440.0 MHz)
PD1: 6
PD2: 5

```

这也可作为配置 SDK 应用程序的 CMake 输出：

```
$./vcocalc.py 48 --cmake
请求频率: 48.0 MHz
实现频率: 48.0 MHz
REFDIV: 1
FBDIV: 120 (VCO = 1440.0 MHz)
PD1: 6
PD2: 5

target_compile_definitions(<program> PRIVATE
 PLL_SYS_REFDIV=1
 PLL_SYS_VCO_FREQ_HZ=1440000000
 PLL_SYS_POSTDIV1=6
 PLL_SYS_POSTDIV2=5
)
```

您还可以传递`--cmake-only`参数仅获取 CMake 输出，使用`--cmake-executable-name`参数将 `<program>` 替换为您正在配置的目标程序名称。

以下示例使用该脚本请求一个功耗最低的 48 MHz 输出：

```
$./vcocalc.py -l 48
请求频率: 48.0 MHz
实现频率: 48.0 MHz
REFDIV: 1
FBDIV: 64 (VCO = 768.0 MHz)
PD1: 4
PD2: 4
```

以下示例使用脚本请求 125 MHz 输出，以实现最低功耗，参考除数 `REFDIV` 固定为 1。尽管我们表示偏好较低的 VCO 频率，但最终频率仍然相当高：

```
$./vcocalc.py -l 125 --lock-refdiv=1
请求频率: 125.0 MHz
实际频率: 125.0 MHz
REFDIV: 1
FBDIV: 125 (VCO = 1500.0 MHz)
PD1: 6
PD2: 2
```

当您的请求输出所需的最佳匹配产生高 VCO 频率时，会出现这种情况。脚本始终返回最佳匹配，仅当存在多个同等优良的匹配时，才偏好较低的 VCO 频率。

您可以通过限制 VCO 频率的上限来规避此问题。以下示例使用脚本请求 125 MHz 系统时钟，限制搜索 VCO 频率低于 800 MHz。由于无精确匹配，脚本会考虑接近（但非精确）的频率匹配。放宽搜索条件以允许附近的非精确匹配，与前例相比，显著降低了最低 VCO 频率：

```
$./vcocalc.py -l 125 --lock-refdiv=1 --vco-max=800
将REFDIV锁定为1
请求频率: 125.0 MHz
达到频率: 126.0 MHz
REFDIV: 1
FBDIV: 63 (VCO = 756.0 MHz)
PD1: 6
```

PD2 : 1

126 MHz的系统时钟相较于目标125 MHz可能为可容忍的偏差，且PLL在生成此时钟时功耗更低。

默认情况下，脚本也会搜索参考分频器，这可能使输出更接近期望值，或允许更高或更低的VCO频率（视偏好而定）。下例允许脚本搜索**FBDIV**值：

```
$./vcocalc.py -l 125
请求频率: 125.0 MHz
实际频率: 125.0 MHz
REFDIV: 2
FBDIV: 125 (VCO = 750.0 MHz)
PD1: 6
PD2: 1
```

这需要非默认的REFDIV值。

请将以下内容添加到您的 CMakeLists.txt 文件中以应用 REFDIV：

```
target_compile_definitions(<program> PRIVATE
 PLL_SYS_REFDIV=2
 PLL_SYS_VCO_FREQ_HZ=750000000
 PLL_SYS_POSTDIV1=6
 PLL_SYS_POSTDIV2=1
)
```

此方法将找到一个解决方案，输出精确符合请求值，且VCO频率恰为最低750 MHz。

[上述内容均假设12 MHz晶体振荡器。RP2350支持的XOSC频率范围详见第8.2节“晶体振荡器（XOSC）”。](#)假设我们使用32 MHz晶体，并需150 MHz系统时钟频率，这是RP2350支持的最大值。您可通过 `--input` 或 `-i` 参数指定输入频率，示例如下：

```
$./vcocalc.py 150 -i 32
请求频率: 150.0 MHz
实际频率: 150.0 MHz
REFDIV: 2
FBDIV: 75 (VCO = 1200.0 MHz)
PD1: 4
PD2: 2
```

这需要非默认的REFDIV值。

请将以下内容添加到您的 CMakeLists.txt 文件中以应用 REFDIV：

```
target_compile_definitions(<program> PRIVATE
 PLL_SYS_REFDIV=2
 PLL_SYS_VCO_FREQ_HZ=1200000000
 PLL_SYS_POSTDIV1=4
 PLL_SYS_POSTDIV2=2
)
```

## 8.6.4. 配置

该SDK使用以下PLL设置：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_clocks/include/hardware/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/include/hardware/clocks.h) 第143至164行

```

143 // RP 系列微控制器中包含两个 PLL：
144 // 1. 'SYS PLL' 生成系统时钟，其频率由 `SYS_CLK_KHZ` 定义。
145 // 2. 'USB PLL' 生成 USB 时钟，其频率由 `USB_CLK_KHZ` 定义。
146 //
147 // 两个 PLL 直接使用晶振输出作为参考频率输入；
// PLL 的参考
148 // 频率无法被时钟模块中的分频器降低。晶振频率由 `XOSC_HZ`（或 149 // `XOSC_KHZ` 或 `XOSC_MHZ`）定义
//
150 //
151 // 对于上述频率，系统的默认定义在12MHz 晶体频率下是准确的
152 // 晶体频率。如需其他频率，必须在
153 // 板级配置文件中定义相应频率及修改后的PLL 设置
154 // 更改任一频率后，请使用 `vcocalc.py` 校验及计算新的PLL设置。

155 //
156 // RP2040 默认PLL 配置：
157 // REF FBDIV VCO POSTDIV
158 // PLL SYS: 12 / 1 = 12MHz * 125 = 1500MHz / 6 / 2 = 125MHz
159 // PLL USB: 12 / 1 = 12MHz * 100 = 1200MHz / 5 / 5 = 48MHz
160 //
161 // RP2350 默认PLL 配置：
162 // REF FBDIV VCO POSTDIV
163 // PLL SYS: 12 / 1 = 12MHz * 125 = 1500MHz / 5 / 2 = 150MHz
164 // PLL USB: 12 / 1 = 12MHz * 100 = 1200MHz / 5 / 5 = 48MHz

```

SDK 中的 `pll_init` 函数（如下所示）会在尝试配置 PLL 之前断言所有条件均为真。

SDK 将 PLL 控制寄存器定义为结构体。随后，它将该结构映射到每个 PLL 实例的内存中。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware\\_structs/include/hardware/structs/pll.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware_structs/include/hardware/structs/pll.h) 第 27 至 74 行

```

27 typedef struct {
28 _REG_(PLL_CS_OFFSET) // PLL_CS
29 // 控制与状态
30 // 0x80000000 [31] LOCK (0) PLL 已锁定
31 // 0x40000000 [30] LOCK_N (0) PLL 未锁定 +
32 // 0x00000100 [8] BYPASS (0) 使用参考时钟直接传递到输出
 该...
33 // 0x0000003f [5:0] REFDIV (0x01) 除以 PLL 输入参考时钟
34 io_rw_32 cs;
35
36 _REG_(PLL_PWR_OFFSET) // PLL_PWR
37 // 控制PLL电源模式
38 // 0x00000020 [5] VCOPD (1) PLL VCO掉电 +
39 // 0x00000008 [3] POSTDIVPD (1) PLL 后置分频器掉电 +
40 // 0x00000004 [2] DSMPD (1) PLL DSM掉电 +
41 // 0x00000001 [0] PD (1) PLL掉电 +
42 io_rw_32 pwr;
43
44 _REG_(PLL_FBDIV_INT_OFFSET) // PLL_FBDIV_INT
45 // 反馈分频器
46 // 0x00000fff [11:0] FB DIV_INT (0x000) 具体约束详见控制寄存器说明
47 io_rw_32 fbdiv_int;
48
49 _REG_(PLL_PRIM_OFFSET) // PLL_PRIM
50 // 控制主输出的PLL后分频器
51 // 0x00070000 [18:16] POSTDIV1 (0x7) 除以 1-7
52 // 0x00007000 [14:12] POSTDIV2 (0x7) 除以 1-7

```

```

53 io_rw_32 prim;
54
55 _REG_(PLL_INTR_OFFSET) // PLL_INTR
56 // 原始中断
57 // 0x00000001 [0] LOCK_N_STICKY (0)
58 io_rw_32 intr;
59
60 _REG_(PLL_INTE_OFFSET) // PLL_INTE
61 // 中断使能
62 // 0x00000001 [0] LOCK_N_STICKY (0)
63 io_rw_32 inte;
64
65 _REG_(PLL_INTF_OFFSET) // PLL_INTF
66 // 中断强制
67 // 0x00000001 [0] LOCK_N_STICKY (0)
68 io_rw_32 intf;
69
70 _REG_(PLL_INTS_OFFSET) // PLL_INTS
71 // 掩码与强制后的中断状态
72 // 0x00000001 [0] LOCK_N_STICKY (0)
73 io_ro_32 ints;
74 } pll_hw_t;

```

SDK 定义了 `pll_init`，用于配置或重新配置 PLL。该函数首先清除 PLL 之前的电源状态，然后计算适当的反馈分频值。有断言用于验证这些值是否满足上述约束条件。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_pll/pll.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_pll/pll.c), 第13至21行

```

13 void pll_init(PLL pll, uint refdiv, uint vco_freq, uint post_div1, uint post_div2) {
14 uint32_t ref_freq = XOSC_HZ / refdiv;
15
16 // 检查 VCO 频率是否处于允许范围内
17 assert(vco_freq >= PICO_PLL_VCO_MIN_FREQ_HZ && vco_freq <= PICO_PLL_VCO_MAX_FREQ_HZ);
18
19 // 计算将参考时钟乘以多少倍得到 VCO 频率
20 // (寄存器命名为 div，是因为通过分频 VCO 输出后与参考时钟比较)
21
22 uint32_t fbdv = vco_freq / ref_freq;

```

PLL 的编程顺序如下：

1. 编程参考时钟分频器（RP2350 情况下为除以 1）。
2. 编程反馈分频器。
3. 开启主电源和压控振荡器（VCO）。
4. 等待 VCO 达到稳定频率，由 `LOCK` 状态标志指示。
5. 设置后置分频器并启动。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_pll/pll.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_pll/pll.c) 第 42 至 69 行

```

42 if ((pll->cs & PLL_CS_LOCK_BITS) &&
43 (refdiv == (pll->cs & PLL_CS_REFDIV_BITS)) &&
44 (fbdv == (pll->fbdv_int & PLL_FBDIV_INT_BITS)) &&
45 (pdiv == (pll->prim & (PLL_PRIM_POSTDIV1_BITS | PLL_PRIM_POSTDIV2_BITS)))) {
46 // 不要干扰已正确配置并运行中的 PLL
47 return;
48 }
49

```

```

50 reset_unreset_block_num_wait_blocking(PLL_RESET_NUM(pll));
51
52 // 在启动 VCO 之前加载与 VCO 相关的分频器
53 pll->cs = refdiv;
54 pll->fbdiv_int = fbdiv;
55
56 // 启动 PLL
57 uint32_t power = PLL_PWR_PD_BITS | // 主电源
58 PLL_PWR_VCOPD_BITS; // VCO 电源
59
60 hw_clear_bits(&pll->pwr, power);
61
62 // 等待 PLL 锁定
63 while (!(pll->cs & PLL_CS_LOCK_BITS)) tight_loop_contents();
64
65 // 设置后置分频器
66 pll->prim = pddiv;
67
68 // 启用后置分频器
69 hw_clear_bits(&pll->pwr, PLL_PWR_POSTDIVPD_BITS);

```

VCO 先启动，随后开启后置分频器，因此在等待 VCO 锁定期问，PLL 不会输出杂讯时钟。

### 8.6.5. 寄存器列表

PLL\_SYS 和 PLL\_USB 寄存器的基址分别为 `0x40050000` 和 `0x40058000`（在 SDK 中定义为 `PLL_SYS_BASE` 和 `PLL_USB_BASE`）。

表 636. PLL 寄存器列表

偏移量	名称	说明
0x00	CS	控制与状态
0x04	PWR	控制 PLL 电源模式。
0x08	FBDIV_INT	反馈除数
0x0c	PRIM	控制主输出的 PLL 后置分频器
0x10	中断	原始中断
0x14	INTE	中断使能
0x18	INTF	中断触发
0x1c	INTS	掩码与强制后的中断状态

#### PLL：CS 寄存器

偏移: 0x00

##### 描述

控制与状态

通用约束条件：

参考时钟频率 最小值=5MHz, 最大值=800MHz

反馈除数 最小值=16, 最大值=320

VCO 频率 最小值=400MHz, 最大值=1600MHz

表 637. CS 寄存器

位	描述	类型	复位
31	LOCK: PLL 已锁定	只读	0x0

位	描述	类型	复位
30	<b>LOCK_N</b> : PLL 未锁定 理论上, 当检测到 PLL 锁定时该位会被清除, 正常情况下不应设置	WC	0x0
29:9	保留。	-	-
8	<b>BYPASS</b> : 将参考时钟直接传递至输出, 而非分频后的VCO。 VCO持续运行, 用户可在参考时钟和分频后的VCO之间切换, 但切换时输出会产生瞬态异常。	读写	0x0
7:6	保留。	-	-
5:0	<b>REFDIV</b> : 对PLL输入的参考时钟进行分频。 当div=0时, 行为未定义。 PLL输出在refdiv变更期间不可预测, 使用前请等待lock=1。	读写	0x01

## PLL: PWR寄存器

偏移: 0x04

### 描述

控制 PLL 电源模式。

表638. PWR  
寄存器

位	描述	类型	复位
31:6	保留。	-	-
5	<b>VCOPD</b> : PLL VCO断电控制 当不需要PLL输出或bypass=1时, 设为高电平以节省功耗。	读写	0x1
4	保留。	-	-
3	<b>POSTDIVPD</b> : PLL后置分频器断电控制 当不需要PLL输出或bypass=1时, 设为高电平以节省功耗。	读写	0x1
2	<b>DSMPD</b> : PLL DSM断电控制 将其置低无任何效果。	读写	0x1
1	保留。	-	-
0	<b>PD</b> : PLL断电控制 当不需要PLL输出时, 设为高电平以节省功耗。	读写	0x1

## PLL: FBDIV\_INT寄存器

偏移: 0x08

### 描述

反馈分频器

(注: 此PLL不支持分数分频)

表639. FBDIV\_INT  
寄存器

位	描述	类型	复位
31:12	保留。	-	-
11:0	有关约束, 请参见 ctrl 寄存器说明	读写	0x000

## PLL: PRIM 寄存器

偏移: 0x0c

**描述**

控制主输出的 PLL 后置分频器  
 (注：该 PLL 不具备次级输出)  
 主输出信号由 VCO 除以 postdiv1\*postdiv2 得出

表640. PRIM 寄存器

位	描述	类型	复位
31:19	保留。	-	-
18:16	<b>POSTDIV1</b> : 除以1至7	读写	0x7
15	保留。	-	-
14:12	<b>POSTDIV2</b> : 除以1至7	读写	0x7
11:0	保留。	-	-

**PLL：INTR 寄存器**

偏移：0x10

**说明**

原始中断

表641. INTR 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>LOCK_N_STICKY</b>	WC	0x0

**PLL：INTE 寄存器**

偏移：0x14

**说明**

中断使能

表642. INTE 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>LOCK_N_STICKY</b>	读写	0x0

**PLL：INTF 寄存器**

偏移：0x18

**说明**

中断强制

表643. INTF 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>LOCK_N_STICKY</b>	读写	0x0

**PLL：INTS 寄存器**

偏移：0x1c

**说明**

掩码与强制后的中断状态

表644. INTS  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>LOCK_N_STICKY</b>	只读	0x0

# 第9章. GPIO

## ⚠ 警告

在某些情况下，下拉功能无法达到预期效果。更多信息，请参见RP2350-E9。

## 9.1. 概述

RP2350配备最多54个多功能通用输入/输出（GPIO）引脚，分为两个组别：

### 存储区0

QFN-60封装（RP2350A）中有30个用户GPIO，或QFN-80封装中有48个用户GPIO

### 存储区1

六个QSPI IO引脚及USB DP/DM引脚

您可以通过处理器上运行的软件，或其他多个功能模块对每个GPIO进行控制。为满足USB信号上升和下降时间的规格要求，USB引脚的模拟特性与GPIO焊盘不同。因此，我们未将其计入54个GPIO的总数。然而，您仍可通过单周期IO子系统（SIO）将其用作UART、I2C或处理器控制的GPIO。

在典型应用场景中，QSPI IO用于从外部闪存执行代码，剩余30个或48个Bank 0 GPIO可供开发者使用。当芯片通过内部OTP引导，或在IO扩展应用中通过SWD进行外部控制时，QSPI引脚可能可用作通用引脚。

所有GPIO均支持数字输入和输出。部分Bank 0 GPIO亦可用作芯片模拟-数字转换器（ADC）的输入：

- QFN-60封装中GPIO 26至29（共计四个）
- QFN-80封装中GPIO 40至47（共计八个）

Bank 0支持以下功能：

- [通过SIO的软件控制](#) —— 第3.1.3节，“GPIO控制”
- 可编程输入输出（PIO）—— 第11章， [PIO](#)
- 2 × SPI—— 第12.3节，“SPI”
- 2 × UART—— 第12.1节，“UART”
- 2 × I2C（双线串行接口）—— 第12.2节，“I2C”
- 8 × QFN-60封装中为双通道PWM，QFN-80封装中为12通道—— 第12.5节，“PWM”
- 2 × [外部时钟输入](#) —— 第8.1.2.4节，“外部时钟”
- 4 × [通用时钟输出](#) —— 第8.1节，“概述”
- 4 × [QFN-60封装中的ADC输入](#)，或[QFN-80封装中的8×输入](#) —— 第12.4节，“ADC和温度传感器”
- 1 × HSTX高速接口 —— 第12.11节，“HSTX”
- 1 × [辅助QSPI芯片选择](#)，用于第二个XIP设备 —— 第12.14节，“QSPI存储接口（QMI）”
- CoreSight执行跟踪输出 —— 第3.5.7节，“跟踪”
- USB VBUS管理 —— 第12.7.3.10节，“VBUS控制”
- 外部中断请求，电平触发或边沿触发 —— 第9.5节，“中断”

Bank 1包含QSPI和USB DP/DM引脚，支持以下功能：

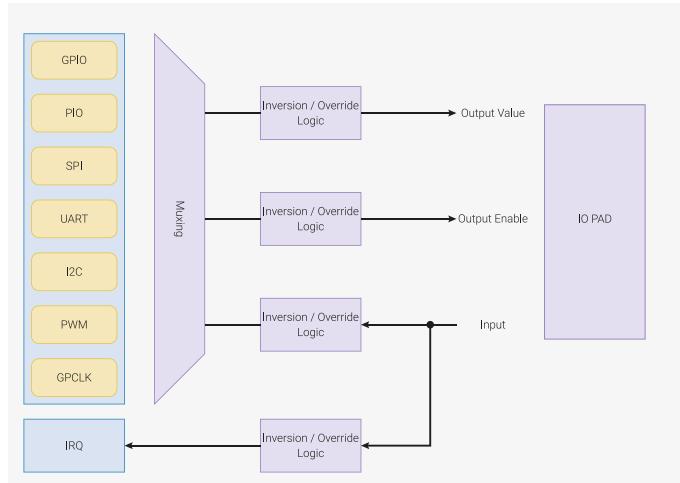
- 通过SIO的软件控制 —— 第3.1.3节，“GPIO控制”
- 通过QSPI存储接口（QMI）实现闪存原地执行（第4.4节，“外部闪存和PSRAM（XIP）”） — 第12.14节，“QSPI存储接口（QMI）”
- UART — 第12.1节，“UART”
- I2C（双线串行接口） — 第12.2节，“I2C”

示例IO的逻辑结构如图41所示。

图41。GPIO的逻辑结构。

每个GPIO可由多个外设之一控制，或由SIO中的软件控制寄存器控制。

功能选择（FSEL）决定哪个外设的输出控制GPIO的方向和输出电平，以及哪个外设输入能读取该GPIO的输入电平。这三种信号（输出电平、输出使能、输入电平）也可通过GPIO控制寄存器进行反向或强制高低电平处理。



## 9.2. 相较于 RP2040 的变化

RP2350 GPIO与RP2040的主要差异如下：

- QFN-80封装中增加18个GPIO
- GPIO功能中增加第三个PIO
- USB DP/DM引脚可用作GPIO
- 在引脚寄存器中增加隔离寄存器（低功耗状态下保持引脚状态，上电后由软件清除）
- 更改了引脚控制的默认复位状态
- GPIO 支持安全访问和非安全访问（详见第10.6节）
- GPIO 中断数量加倍，以区分安全和非安全访问
- 新增中断汇总寄存器，便于快速检查哪些 GPIO 存在待处理中断

## 9.3. 复位状态

首次上电时，Bank 0 IO（QFN-60 封装中 GPIO 0 至 29，QFN-80 封装中 GPIO 0 至 47）处于以下状态：

- 输出缓冲器呈高阻抗状态
- 输入缓冲器被禁用
- 下拉状态
- 隔离锁存器设置为锁存状态（详见第9.7节）

每个引脚的输出禁用位（GPIO0.OD）在复位时为清零，但 IO 复用重置为无功能状态，

以确保输出缓冲器处于高阻抗状态。

### ！重要

引脚复位状态与 RP2040 不同，后者仅在 GPIO 26 至 29（自 B2 版本起）禁用数字输入，且无隔离锁存器。应用程序必须启用 pad 输入（GPIO0.IE = 1）并禁用 pad 隔离锁存器（GPIO0.ISO = 0），方可将 pads 用于数字 I/O。SDK 函数 `gpio_set_function()` 会自动完成此类操作。

Bank 1 的 IO 复位状态与 Bank 0 的 GPIO 相同，唯输入使能（IE）复位为 1，且上下拉状态存在差异：SCK、SD0 和 SD1 为下拉，SD2、SD3 与 CSn 为上拉。

### ！注意

若要将 Bank 0 GPIO 用作第二个片选，需外接上拉电阻，以确保第二个 QSPI 设备在片选未被激活时不会上电。

pads 在任一以下情况下将恢复至复位状态：

- 欠压（褐化）复位
- 将 RUN 引脚拉低
- 通过 SWD 设置 SW-DP 的 CDBG\_RSTREQ
- 通过 SWD 设置 RP-AP 救援复位

如果引脚的隔离锁存器处于锁存状态（第9.7节），那么重置PADS和IO寄存器不会将该引脚物理恢复到复位状态。隔离锁存器防止上游信号传播至引脚。

清除ISO位以允许信号传播。

## 9.4. 功能选择

要为GPIO分配功能，请写入相应引脚的 `CTRL` 寄存器中的`FUNCSEL`字段。有关GPIO及对应寄存器的列表，请参见表645。示例请参见`GPIO0_CTRL`。表中列出功能的说明详见表646。

每个GPIO一次只能选择一项功能。每个外设输入（如UART0 RX）应一次仅被一个GPIO选择。若将相同外设输入连接至多个GPIO，外设将接收到这些GPIO输入的逻辑或。

表645。通用输入/输出 (GPIO) 第0组功能

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
0		SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB 过流检测	
1		SPI0 片选 (CSn)	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM0 B	SIO	PIO0	PIO1	PIO2	TRACECLK	USB VBUS 检测	
2		SPI0 时钟 (SCK)	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM1 A	SIO	PIO0	PIO1	PIO2	TRACEDATA0	USB VBUS 使能	UART0 TX
3		SPI0 发送 (TX)	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM1 B	SIO	PIO0	PIO1	PIO2	TRACEDATA1	USB 过流检测	UART0 接收 (RX)
4		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	TRACEDATA2	USB VBUS 检测	
5		SPI0 片选 (CSn)	UART1 RX	I2C0 时钟线 (SCL)	PWM2 B	SIO	PIO0	PIO1	PIO2	TRACEDATA3	USB VBUS 使能	
6		SPI0 时钟 (SCK)	UART1 CTS	I2C1 数据线 (SDA)	PWM3 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 TX
7		SPI0 发送 (TX)	UART1 RTS	I2C1 时钟线 (SCL)	PWM3 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART1 RX
8		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS 使能	
9		SPI1 CSn	UART1 RX	I2C0 时钟线 (SCL)	PWM4 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	
10		SPI1 SCK	UART1 CTS	I2C1 数据线 (SDA)	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART1 TX
11		SPI1 TX	UART1 RTS	I2C1 时钟线 (SCL)	PWM5 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART1 RX
12	HSTX	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB 过流检测	
13	HSTX	SPI1 CSn	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM6 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB VBUS 检测	
14	HSTX	SPI1 SCK	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM7 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS 使能	UART0 TX
15	HSTX	SPI1 TX	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM7 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB 过流检测	UART0 接收 (RX)
16	HSTX	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
17	HSTX	SPI0 片选 (CSn)	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM0 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
18	HSTX	SPI0 时钟 (SCK)	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM1 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART0 TX
19	HSTX	SPI0 发送 (TX)	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM1 B	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS 检测	UART0 接收 (RX)
20		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB VBUS 使能	
21		SPI0 片选 (CSn)	UART1 RX	I2C0 时钟线 (SCL)	PWM2 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB 过流检测	
22		SPI0 时钟 (SCK)	UART1 CTS	I2C1 数据线 (SDA)	PWM3 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIN1	USB VBUS 检测	UART1 TX

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
23		SPI0 发送 (TX)	UART1 RTS	I2C1 时钟线 (SCL)	PWM3 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB VBUS 使能	UART1 RX
24		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT2	USB 过流检测	
25		SPI1 CSn	UART1 RX	I2C0 时钟线 (SCL)	PWM4 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT3	USB VBUS 检测	
26		SPI1 SCK	UART1 CTS	I2C1 数据线 (SDA)	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART1 TX
27		SPI1 TX	UART1 RTS	I2C1 时钟线 (SCL)	PWM5 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 RX
28		SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
29		SPI1 CSn	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM6 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
<b>GPIO 30 至 47 仅限 QFN-80 型:</b>												
30		SPI1 SCK	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM7 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART0 TX
31		SPI1 TX	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM7 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART0 接收 (RX)
32		SPI0 RX	UART0 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
33		SPI0 片选 (CSn)	UART0 接收 (RX)	I2C0 时钟线 (SCL)	PWM8 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	
34		SPI0 时钟 (SCK)	UART0 清除发送 (CTS)	I2C1 数据线 (SDA)	PWM9 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART0 TX
35		SPI0 发送 (TX)	UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)	PWM9 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART0 接收 (RX)
36		SPI0 RX	UART1 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	
37		SPI0 片选 (CSn)	UART1 RX	I2C0 时钟线 (SCL)	PWM10 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
38		SPI0 时钟 (SCK)	UART1 CTS	I2C1 数据线 (SDA)	PWM11 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	UART1 TX
39		SPI0 发送 (TX)	UART1 RTS	I2C1 时钟线 (SCL)	PWM11 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 RX
40		SPI1 RX	UART1 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	
41		SPI1 CSn	UART1 RX	I2C0 时钟线 (SCL)	PWM8 B	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	
42		SPI1 SCK	UART1 CTS	I2C1 数据线 (SDA)	PWM9 A	SIO	PIO0	PIO1	PIO2		USB 过流检测	UART1 TX
43		SPI1 TX	UART1 RTS	I2C1 时钟线 (SCL)	PWM9 B	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART1 RX
44		SPI1 RX	UART0 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2		USB VBUS 使能	

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
45		SPI1 CSn	UART0 接收 (RX)	I <sub>2</sub> C0 时钟线 (SCL)	PWM10 B	SIO	PIO0	PIO1	PIO2		USB 过流检测	
46		SPI1 SCK	UART0 清除发送 (CTS)	I <sub>2</sub> C1 数据线 (SDA)	PWM11 A	SIO	PIO0	PIO1	PIO2		USB VBUS 检测	UART0 TX
47		SPI1 TX	UART0 请求发送 (RTS)	I <sub>2</sub> C1 时钟线 (SCL)	PWM11 B	SIO	PIO0	PIO1	PIO2	QMI CS1n	USB VBUS 使能	UART0 接收 (RX)

表646。GPIO 用户  
Bank 功能  
说明

功能名称	描述
SPIx	将内部 PL022 SPI 外设之一连接至 GPIO。
UARTx	将内部 PL011 UART 外设之一连接至 GPIO。
I2Cx	将内部 DW I2C 外设之一连接至 GPIO。
PWMx A/B	将 PWM 片段连接至 GPIO。共有十二个 PWM 片段，每个包含两个输出通道（A/B）。B 引脚亦可用作输入，用于频率及占空比测量。
SIO	通过单周期输入输出单元（SIO）对 GPIO 进行软件控制。处理器驱动 GPIO 时必须选择 SIO 功能（ <a href="#">F5</a> ），但输入端始终连接，因而软件可随时检测 GPIO 状态。
PIOx	将可编程输入输出模块（PIO）之一连接至 GPIO。PIO 能实现多种接口，且配备内部管脚映射硬件，允许数字接口灵活分布于 Bank 0 GPIO 上。必须选择 PIO 功能（ <a href="#">F6</a> 、 <a href="#">F7</a> 、 <a href="#">F8</a> ）以使 PIO 驱动 GPIO，但输入始终连接，因此 PIO 始终可检测所有管脚状态。
HSTX	将高速传输外设（HSTX）连接至 GPIO。
时钟 GPINx	通用时钟输入。可路由至 RP2350 的多个内部时钟域，例如为 AON 定时器提供 1Hz 时钟，或连接至内部频率计数器。
时钟 GPOUTx	通用时钟输出。可将多个内部时钟（包括 PLL 输出）驱动至 GPIO，支持可选的整数分频。
TRACECLK, TRACEDATAx	来自 Cortex-M33 处理器（仅限 Arm 架构）的 CoreSight 执行跟踪输出。
USB 过流检测 / VBUS 电源线 检测 / VBUS 电源线使能	USB 电源控制信号与内部 USB 控制器间的连接。
QMI CS1n	QSPI 总线辅助芯片选择，用以支持从附加闪存或 PSRAM 设备即取即执行。

Bank 1 功能选择的操作与 Bank 0 完全相同，但其寄存器位于不同的寄存器块，起始地址为 [USBPHY\\_DP\\_CTRL](#)。

表647。GPIO Bank  
1功能

引脚	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
USB DP			UART1 TX	I2C0 SDA		SIO						
USB DM			UART1 RX	I2C0 时钟线 (SCL)		SIO						
QSPI SCK	QMI SCK		UART1 CTS	I2C1 数据线 (SDA)		SIO						UART1 TX
QSPI CSn	QMI CS0n		UART1 RTS	I2C1 时钟线 (SCL)		SIO						UART1 RX
QSPI SD0	QMI SD0		UART0 TX	I2C0 SDA		SIO						
QSPI SD1	QMI SD1		UART0 接收 (RX)	I2C0 时钟线 (SCL)		SIO						
QSPI SD2	QMI SD2		UART0 清除发送 (CTS)	I2C1 数据线 (SDA)		SIO						UART0 TX
QSPI SD3	QMI SD3		UART0 请求发送 (RTS)	I2C1 时钟线 (SCL)		SIO						UART0 接收 (RX)

表648。GPIO Bank  
1功能  
描述

功能名称	描述
UARTx	将内部 PL011 UART 外设之一连接至 GPIO。
I2Cx	将内部 DW I2C 外设之一连接至 GPIO。

功能名称	描述
SIO	通过单周期 IO (SIO) 模块进行 GPIO 的软件控制。处理器驱动 GPIO 时必须选择 SIO 功能 ( <a href="#">F5</a> )，但输入端始终连接，因而软件可随时检测 GPIO 状态。
QMI	QSPI 存储器接口外设，用于从外部 QSPI 闪存或 PSRAM 存储器设备执行就地操作。

这六个QSPI Bank GPIO引脚通常由XIP外设用以与外部闪存设备通信。

然而，存在两种情形允许将这些引脚用作软件控制的GPIO：

- 若SPI或Dual-SPI闪存设备用于执行就地操作，则SD2和SD3引脚不用于闪存访问，可用于电路板上的其他GPIO功能。
- 若RP2350采用无闪存配置（仅支持USB及OTP启动），则所有六个引脚均可用作软件控制的GPIO。

## 9.5. 中断

每个GPIO引脚在以下四种情形下均可生成中断：

- 高电平：GPIO引脚逻辑值为1
- 低电平：GPIO引脚逻辑值为0
- 上升沿：GPIO已从逻辑0变为逻辑1
- 下降沿：GPIO已从逻辑1变为逻辑0

电平中断未被锁存。这意味着，如果引脚处于逻辑1且电平高中断处于激活状态，一旦引脚变为逻辑0，该中断将立即失效。边沿中断存储于 [INTR](#) 寄存器中，可通过写入 [INTR](#) 寄存器进行清除。

针对三个中断目标：proc 0、proc 1及dormant\_wake，设有使能、状态及强制寄存器。proc 0的寄存器为使能（PROCO\_INTE0）、状态（PROCO\_INTS0）和强制（PROCO\_INTF0）。Dormant wake用于将ROSC或XOSC从休眠模式唤醒。有关休眠模式的详细信息，请参见第6.5.6.2节。

每个 IO 银行、中断请求目标和安全域的组合均对应一个中断输出。此类输出总计十二个：

- IO 银行 0 到休眠唤醒（安全域和非安全域）
- IO 银行 0 到处理器 0（安全域和非安全域）
- IO 银行 0 到处理器 1（安全域和非安全域）
- IO QSPI 到休眠唤醒（安全域和非安全域）
- IO QSPI 到处理器 0（安全域和非安全域）
- IO QSPI 到处理器 1（安全域和非安全域）

每个中断输出均拥有独立的使能寄存器数组（INTE），用于配置何种 GPIO 事件会触发中断。当至少发生一个已使能事件时，中断即被触发；当所有已使能事件均通过相应 INTR 寄存器确认后，中断将被撤销。

这意味着用户能够同时监控多个 GPIO 事件。

汇总寄存器可用于快速检查待处理的 GPIO 中断。示例请参见 IRQSUMMARY\_PROC0\_NONSECURE0。

## 9.6. 引脚

### ⚠ 警告

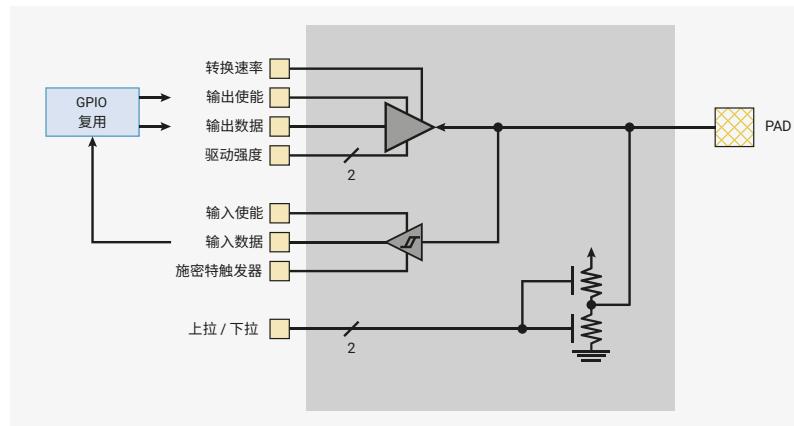
在某些情况下，下拉功能无法达到预期效果。更多信息，请参见RP2350-E9。

每个GPIO均通过一个 **pad**与芯片外部连接。Pad是芯片内部逻辑与外部电路之间的电气接口。它们转换信号电压等级，支持更高电流，并提供对静电放电（ESD）事件的部分保护。您可以通过以下方式调整pad的电气性能，以满足外部电路的要求：

- 输出驱动强度可设置为2mA、4mA、8mA或12mA。
- 输出转换速率可设置为慢速或快速。
- 可启用输入迟滞（施密特触发模式）。
- 可启用上拉或下拉电阻，以在输出驱动器禁用时设置输出信号电平。
- 可禁用输入缓冲区，以在pad未使用、未连接或连接至模拟信号时降低电流消耗。

图42展示了pad的一个示例。

图42。单个IO引脚示意图



该引脚的输出使能、输出数据和输入数据端口通过IO复用器连接至控制该引脚的功能模块。

其他所有端口均由引脚控制寄存器控制。可通过该寄存器覆盖控制引脚功能的输出使能信号，从而禁用引脚的输出驱动器。有关引脚控制寄存器的示例，请参见GPIO0。

引脚的输出信号电平与输入信号的可接受电平均由数字IO电源（IOVDD）决定。

IOVDD可为1.8V至3.3V之间任一标称电压，但为满足1.8V供电条件下的规格，须通过向该引脚的VOLTAGE\_SELECT寄存器写入1来调整输入阈值。默认情况下，引脚输入阈值适用于IOVDD电压范围为2.5V至3.3V。使用1.8V电压配合默认输入阈值可确保安全运行，但输入阈值将不符合规格要求。

### ⚠ 警告

使用高于1.8V的IOVDD电压，而输入阈值设置为1.8V，可能会导致芯片损坏。

垫输入阈值按银行单独调整，分别对应用户IO银行（IO银行0）和QSPI IO银行的VOLTAGE\_SELECT寄存器。但两个银行共用相同的数字IO电源（IOVDD），因此两个寄存器应始终设置为相同的值。

垫寄存器详情见第9.11.3节“垫控制—用户银行”及第9.11.4节“垫控制—QSPI银行”。

### 9.6.1. 总线保持器模式

每个垫在任何时刻仅能启用上拉电阻或下拉电阻之一。不可能同时启用二者。相反，如果同时设置GPIO0.PDE和GPIO0.PUE位，将启用总线保持器模式，此时垫状态为：

- 当其输入为高电平时，输出被拉高。
- 当其输入为低电平时，输出被拉低。

当输出缓冲器被禁用且引脚未被任何外部源驱动时，此模式会弱保持引脚当前的逻辑状态。该引脚不会漂移至中间电平。

总线保持器模式依赖切换核心域中的控制逻辑，因此在核心断电时无法工作。相反，当总线保持器模式启用且核心断电时，会如第9.7节所述，将当前输出控制（上拉或下拉）锁存于引脚隔离锁存器中。

## 9.7. 引脚隔离锁存器

RP2350具备扩展低功耗状态，允许除POWMAN及部分CoreSight调试逻辑外，所有内部逻辑在软件控制下完全断电。这包括关闭所有外设、IO复用及引脚控制寄存器，但同时也存在引脚信号在进入和退出低功耗状态时发生非预期切换的风险。

为确保引脚状态始终定义明确，所有从切换核心电源域传递至引脚的信号均经过隔离锁存器。在正常操作情况下，锁存器处于透明状态，因此引脚完全由切换核心电源域内的逻辑控制，如UART或处理器。然而，当每个引脚的ISO位被设置（例如GPIO0.ISO）或切换核心电源域断电时，当前传递到该引脚的控制信号将被锁存，直到隔离被禁用。这包括输出使能状态、输出高/低电平以及上拉/下拉电阻使能状态。来自引脚返回至切换核心电源域的输入信号不受隔离。

因此，当切换核心逻辑断电时，所有Bank 0和Bank 1引脚保持断电前的输出状态，除非被POWMAN中的常电逻辑覆盖。切换核心电源域重新上电后，所有GPIO ISO位复位为1，因此断电前的状态将持续保持，直到用户软件启动并清除ISO位，表明已准备重新使用该引脚。未设置IO复用的引脚可保持隔离状态，且可无限期维持断电前的状态。

当软件完成对指定引脚及其所复用的IO复用设置后，应当清除ISO位。此时，隔离锁存器将重新变为透明状态：通过IO复用模块传递的输出信号现已反映于引脚输出状态，外设可由此实现与外界的通信。

此过程允许对切换的核心域进行电源循环，而不会在引脚输出端产生任何可能干扰外部连接硬件正常运行的信号切换。

#### 注意

从RP2040移植的非SDK应用在使用GPIO之前必须清除ISO位，因RP2040不具备该功能。SDK在调用`gpio_set_function()`时会自动清除ISO位。

隔离锁存器由始终通电的电源域复位，具体复位条件包括以下任一项：

- 上电复位
- 欠压复位
- RUN 引脚被拉低
- SW-DP CDBGRSTREQ
- RP-AP 救援复位

锁存器复位为被隔离信号的复位值。例如，在Bank 0的GPIO上，输入使能控制

(GPIO0.IE) 复位为 0 (输入禁用)，因此这些信号的隔离锁存器也被复位为 0。复位隔离锁存器会强制端口即使当前处于隔离状态，仍恢复到其复位状态。

ISO 控制位（例如 GPIO0.ISO）由顶层切换核心域隔离信号复位，该信号由 POWMAN 在切换核心域断电前断言，通电后取消断言。这意味着进入和退出切换核心域断电的睡眠状态后，所有 GPIO 都维持隔离状态；随后您可以逐个重新启用它们。ISO 控制位不会被由 RESETS 控制寄存器驱动的 PADS 寄存器块复位影响：复位 PADS 寄存器块会将非隔离端口恢复到复位状态，但对隔离端口无效。

## 9.8. 处理器 GPIO 控制 (SIO)

单周期IO子系统（第3.1节）包含内存映射的GPIO寄存器。处理器可利用这些寄存器对GPIO执行输入/输出操作：

- GPIO\_OUT 和 GPIO\_HI\_OUT 寄存器用于设置输出电平：1 表示高电平，0 表示低电平。
- GPIO\_OE 和 GPIO\_HI\_OE 寄存器用于设置输出使能：1 表示输出，0 表示输入。
- GPIO\_IN 和 GPIO\_HI\_IN 寄存器用于读取GPIO的输入信号。

所有这些寄存器均为32位。低位寄存器（如 GPIO\_OUT）连接GPIO 0至31，高位寄存器（如 GPIO\_HI\_OUT）连接GPIO 32至47，以及QSPI焊盘和USB DM/DP焊盘。

要使输出及输出使能寄存器生效，必须为每个GPIO选择SIO功能（function 5）。

但即使未选择SIO功能，GPIO输入寄存器仍会返回GPIO的输入值，因此处理器始终能够检测任意引脚的输入状态。

SIO GPIO寄存器在两个处理器之间以及安全和非安全安全域之间共享。此举避免了因从不同上下文选择多个GPIO功能进行访问而引起的编程错误。

非安全代码对SIO寄存器的访问受限于GPIO\_NSmask0和GPIO\_NSmask1中定义的非安全GPIO掩码。对安全GPIO的非安全写入将被忽略。非安全读取安全GPIO将返回0。

这些寄存器在SIO GPIO寄存器部分（第3.1.3节）中有更为详细的文档说明。

DMA无法访问SIO子系统中的寄存器。推荐使用的DMA传输至GPIO的方法是通过PIO程序持续将TX FIFO数据传输至GPIO输出，这相比直接DMA写入GPIO寄存器能提供更为一致的时序。

## 9.9. GPIO 协处理器端口

每个Cortex-M33处理器的协处理器端口 0连接至GPIO协处理器接口。这些协处理器指令为Arm软件提供了快速访问SIO GPIO寄存器的能力：

- 任何SIO GPIO寄存器访问的等效操作均为单条指令，无需预先计算32位寄存器地址
- 对任意单个GPIO的索引写操作即为单条指令
- 单条指令即可读写64位数据

这降低了GPIO访问对周边软件时序的影响，例如在中断处理程序中添加GPIO跟踪以诊断复杂时序问题时。

安全代码与非安全代码均可访问协处理器。非安全代码仅能查看由ACCESSCTRL GPIO\_NSmask0/1定义的GPIO寄存器受限视图。

GPIO协处理器指令详见第3.6.1节。

## 9.10. 软件示例

### 9.10.1. 选择 IO 功能

IO引脚可执行多种功能，使用前必须配置。例如，您可能希望其作为 `UART_TX`引脚，或作为 `PWM`输出。SDK提供了`gpio_set_function`以实现该配置。许多SDK示例通常在早期调用`gpio_set_function`，以启用UART打印。

该SDK首先定义了一个结构体，用以表示IO Bank 0，即用户IO Bank的寄存器。每个IO均包含一个状态寄存器，随后是一个控制寄存器。对于 `N`个IO，SDK实例化包含状态与控制寄存器的结构体，作为 `io[N]`重复 `N`次。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware\\_structs/include/hardware/structs/io\\_bank0.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2350/hardware_structs/include/hardware/structs/io_bank0.h) 第179行至445行

```

179 typedef struct {
180 io_bank0_status_ctrl_hw_t io[48];
181
182 uint32_t _pad0[32];
183
184 // (描述摘自数组索引0寄存器IO_BANK0_IRQSUMMARY_PROC0_SECURE0, 其他数组索引亦适用)
185
186 _REG_(IO_BANK0_IRQSUMMARY_PROC0_SECURE0_OFFSET) // IO_BANK0_IRQSUMMARY_PROC0_SECURE0
187 // 0x80000000 [31] GPIO31 (0)
188 // 0x40000000 [30] GPIO30 (0)
189 // 0x20000000 [29] GPIO29 (0)
190 // 0x10000000 [28] GPIO28 (0)
191 // 0x08000000 [27] GPIO27 (0)
192 // 0x04000000 [26] GPIO26 (0)
193 // 0x02000000 [25] GPIO25 (0)
194 // 0x01000000 [24] GPIO24 (0)
195 // 0x00800000 [23] GPIO23 (0)
196 // 0x00400000 [22] GPIO22 (0)
197 // 0x00200000 [21] GPIO21 (0)
198 // 0x00100000 [20] GPIO20 (0)
199 // 0x00080000 [19] GPIO19 (0)
200 // 0x00040000 [18] GPIO18 (0)
201 // 0x00020000 [17] GPIO17 (0)
202 // 0x00010000 [16] GPIO16 (0)
203 // 0x00008000 [15] GPIO15 (0)
204 // 0x00004000 [14] GPIO14 (0)
205 // 0x00002000 [13] GPIO13 (0)
206 // 0x00001000 [12] GPIO12 (0)
207 // 0x00000800 [11] GPIO11 (0)
208 // 0x00000400 [10] GPIO10 (0)
209 // 0x00000200 [9] GPIO09 (0)
210 // 0x00000100 [8] GPIO08 (0)
211 // 0x00000080 [7] GPIO07 (0)
212 // 0x00000040 [6] GPIO06 (0)
213 // 0x00000020 [5] GPIO05 (0)
214 // 0x00000010 [4] GPIO04 (0)
215 // 0x00000008 [3] GPIO03 (0)
216 // 0x00000004 [2] GPIO02 (0)
217 // 0x00000002 [1] GPIO01 (0)
218 // 0x00000001 [0] GPIO00 (0)
219
220 io_ro_32 irqsummary_proc0_secure[2];
221
222 _REG_(IO_BANK0_IRQSUMMARY_PROC0_NONSECURE0_OFFSET) //
223 IO_BANK0_IRQSUMMARY_PROC0_NONSECURE0
224 // 0x80000000 [31] GPIO31 (0)

```

```

223 // 0x40000000 [30] GPIO030 (0)
224 // 0x20000000 [29] GPIO029 (0)
225 // 0x10000000 [28] GPIO028 (0)
226 // 0x08000000 [27] GPIO027 (0)
227 // 0x04000000 [26] GPIO026 (0)
228 // 0x02000000 [25] GPIO025 (0)
229 // 0x01000000 [24] GPIO024 (0)
230 // 0x00800000 [23] GPIO023 (0)
231 // 0x00400000 [22] GPIO022 (0)
232 // 0x00200000 [21] GPIO021 (0)
233 // 0x00100000 [20] GPIO020 (0)
234 // 0x00080000 [19] GPIO019 (0)
235 // 0x00040000 [18] GPIO018 (0)
236 // 0x00020000 [17] GPIO017 (0)
237 // 0x00010000 [16] GPIO016 (0)
238 // 0x00008000 [15] GPIO015 (0)
239 // 0x00004000 [14] GPIO014 (0)
240 // 0x00002000 [13] GPIO013 (0)
241 // 0x00001000 [12] GPIO012 (0)
242 // 0x00000800 [11] GPIO011 (0)
243 // 0x00000400 [10] GPIO010 (0)
244 // 0x00000200 [9] GPIO09 (0)
245 // 0x00000100 [8] GPIO08 (0)
246 // 0x00000080 [7] GPIO07 (0)
247 // 0x00000040 [6] GPIO06 (0)
248 // 0x00000020 [5] GPIO05 (0)
249 // 0x00000010 [4] GPIO04 (0)
250 // 0x00000008 [3] GPIO03 (0)
251 // 0x00000004 [2] GPIO02 (0)
252 // 0x00000002 [1] GPIO01 (0)
253 // 0x00000001 [0] GPIO00 (0)
254 io_ro_32 irqsummary_proc0_nonscure[2];
255
256 // (该描述摘自数组索引0的寄存器IO_BANK0_IRQSUMMARY_PROC1_SECURE0，适用于其他数组索引)

257 _REG_(IO_BANK0_IRQSUMMARY_PROC1_SECURE0_OFFSET) // IO_BANK0_IRQSUMMARY_PROC1_SECURE0
258 // 0x80000000 [31] GPIO031 (0)
259 // 0x40000000 [30] GPIO030 (0)
260 // 0x20000000 [29] GPIO029 (0)
261 // 0x10000000 [28] GPIO028 (0)
262 // 0x08000000 [27] GPIO027 (0)
263 // 0x04000000 [26] GPIO026 (0)
264 // 0x02000000 [25] GPIO025 (0)
265 // 0x01000000 [24] GPIO024 (0)
266 // 0x00800000 [23] GPIO023 (0)
267 // 0x00400000 [22] GPIO022 (0)
268 // 0x00200000 [21] GPIO021 (0)
269 // 0x00100000 [20] GPIO020 (0)
270 // 0x00080000 [19] GPIO019 (0)
271 // 0x00040000 [18] GPIO018 (0)
272 // 0x00020000 [17] GPIO017 (0)
273 // 0x00010000 [16] GPIO016 (0)
274 // 0x00008000 [15] GPIO015 (0)
275 // 0x00004000 [14] GPIO014 (0)
276 // 0x00002000 [13] GPIO013 (0)
277 // 0x00001000 [12] GPIO012 (0)
278 // 0x00000800 [11] GPIO011 (0)
279 // 0x00000400 [10] GPIO010 (0)
280 // 0x00000200 [9] GPIO09 (0)
281 // 0x00000100 [8] GPIO08 (0)
282 // 0x00000080 [7] GPIO07 (0)
283 // 0x00000040 [6] GPIO06 (0)
284 // 0x00000020 [5] GPIO05 (0)
285 // 0x00000010 [4] GPIO04 (0)

```

```

286 // 0x00000008 [3] GPIO3 (0)
287 // 0x00000004 [2] GPIO2 (0)
288 // 0x00000002 [1] GPIO1 (0)
289 // 0x00000001 [0] GPIO0 (0)
290 io_ro_32 irqsummary_proc1_secure[2];
291
292 // (该描述摘自数组索引0的寄存器IO_BANK0_IRQSUMMARY_PROC1_NONSECURE0，适用于其他数组索引)

293 _REG_(IO_BANK0_IRQSUMMARY_PROC1_NONSECURE0_OFFSET) //
 IO_BANK0_IRQSUMMARY_PROC1_NONSECURE0
294 // 0x80000000 [31] GPIO31 (0)
295 // 0x40000000 [30] GPIO30 (0)
296 // 0x20000000 [29] GPIO29 (0)
297 // 0x10000000 [28] GPIO28 (0)
298 // 0x08000000 [27] GPIO27 (0)
299 // 0x04000000 [26] GPIO26 (0)
300 // 0x02000000 [25] GPIO25 (0)
301 // 0x01000000 [24] GPIO24 (0)
302 // 0x00800000 [23] GPIO23 (0)
303 // 0x00400000 [22] GPIO22 (0)
304 // 0x00200000 [21] GPIO21 (0)
305 // 0x00100000 [20] GPIO20 (0)
306 // 0x00080000 [19] GPIO19 (0)
307 // 0x00040000 [18] GPIO18 (0)
308 // 0x00020000 [17] GPIO17 (0)
309 // 0x00010000 [16] GPIO16 (0)
310 // 0x00008000 [15] GPIO15 (0)
311 // 0x00004000 [14] GPIO14 (0)
312 // 0x00002000 [13] GPIO13 (0)
313 // 0x00001000 [12] GPIO12 (0)
314 // 0x00000800 [11] GPIO11 (0)
315 // 0x00000400 [10] GPIO10 (0)
316 // 0x00000200 [9] GPIO9 (0)
317 // 0x00000100 [8] GPIO8 (0)
318 // 0x00000080 [7] GPIO7 (0)
319 // 0x00000040 [6] GPIO6 (0)
320 // 0x00000020 [5] GPIO5 (0)
321 // 0x00000010 [4] GPIO4 (0)
322 // 0x00000008 [3] GPIO3 (0)
323 // 0x00000004 [2] GPIO2 (0)
324 // 0x00000002 [1] GPIO1 (0)
325 // 0x00000001 [0] GPIO0 (0)
326 io_ro_32 irqsummary_proc1_nonsecure[2];
327
328 // (复制自数组索引0寄存器的描述)
 IO_BANK0_IRQSUMMARY_DORMANT_WAKE_SECURE0对于其他数组索引同样适用)

329 _REG_(IO_BANK0_IRQSUMMARY_DORMANT_WAKE_SECURE0_OFFSET) //
 IO_BANK0_IRQSUMMARY_DORMANT_WAKE_SECURE0
330 // 0x80000000 [31] GPIO31 (0)
331 // 0x40000000 [30] GPIO30 (0)
332 // 0x20000000 [29] GPIO29 (0)
333 // 0x10000000 [28] GPIO28 (0)
334 // 0x08000000 [27] GPIO27 (0)
335 // 0x04000000 [26] GPIO26 (0)
336 // 0x02000000 [25] GPIO25 (0)
337 // 0x01000000 [24] GPIO24 (0)
338 // 0x00800000 [23] GPIO23 (0)
339 // 0x00400000 [22] GPIO22 (0)
340 // 0x00200000 [21] GPIO21 (0)
341 // 0x00100000 [20] GPIO20 (0)
342 // 0x00080000 [19] GPIO19 (0)
343 // 0x00040000 [18] GPIO18 (0)
344 // 0x00020000 [17] GPIO17 (0)
345 // 0x00010000 [16] GPIO16 (0)

```

```

346 // 0x000008000 [15] GPIO15 (0)
347 // 0x000004000 [14] GPIO14 (0)
348 // 0x000002000 [13] GPIO13 (0)
349 // 0x000001000 [12] GPIO12 (0)
350 // 0x000000800 [11] GPIO11 (0)
351 // 0x000000400 [10] GPIO10 (0)
352 // 0x000000200 [9] GPIO09 (0)
353 // 0x000000100 [8] GPIO08 (0)
354 // 0x000000080 [7] GPIO07 (0)
355 // 0x000000040 [6] GPIO06 (0)
356 // 0x000000020 [5] GPIO05 (0)
357 // 0x000000010 [4] GPIO04 (0)
358 // 0x000000008 [3] GPIO03 (0)
359 // 0x000000004 [2] GPIO02 (0)
360 // 0x000000002 [1] GPIO01 (0)
361 // 0x000000001 [0] GPIO00 (0)
362 io_ro_32 irqssummary_dormant_wake_secure[2];
363
364 // (复制自数组索引0寄存器的描述)
365 _REG_(IO_BANK0_IRQSUMMARY_DORMANT_WAKE_NONSECURE0对于其他数组索引同样适用) 365 _REG_(IO_BANK0_
IRQSUMMARY_DORMANT_WAKE_NONSECURE0_OFFSET) // IO_BANK0_IRQSUMMARY_DO
RMANT_WAKE_NONSECURE0
366 // 0x80000000 [31] GPIO031 (0)
367 // 0x40000000 [30] GPIO030 (0)
368 // 0x20000000 [29] GPIO029 (0)
369 // 0x10000000 [28] GPIO028 (0)
370 // 0x08000000 [27] GPIO027 (0)
371 // 0x04000000 [26] GPIO026 (0)
372 // 0x02000000 [25] GPIO025 (0)
373 // 0x01000000 [24] GPIO024 (0)
374 // 0x00800000 [23] GPIO023 (0)
375 // 0x00400000 [22] GPIO022 (0)
376 // 0x00200000 [21] GPIO021 (0)
377 // 0x00100000 [20] GPIO020 (0)
378 // 0x00080000 [19] GPIO019 (0)
379 // 0x00040000 [18] GPIO018 (0)
380 // 0x00020000 [17] GPIO017 (0)
381 // 0x00010000 [16] GPIO016 (0)
382 // 0x00008000 [15] GPIO015 (0)
383 // 0x00004000 [14] GPIO014 (0)
384 // 0x00002000 [13] GPIO013 (0)
385 // 0x00001000 [12] GPIO012 (0)
386 // 0x00000800 [11] GPIO011 (0)
387 // 0x00000400 [10] GPIO010 (0)
388 // 0x00000200 [9] GPIO009 (0)
389 // 0x00000100 [8] GPIO008 (0)
390 // 0x00000080 [7] GPIO007 (0)
391 // 0x00000040 [6] GPIO006 (0)
392 // 0x00000020 [5] GPIO005 (0)
393 // 0x00000010 [4] GPIO004 (0)
394 // 0x00000008 [3] GPIO003 (0)
395 // 0x00000004 [2] GPIO002 (0)
396 // 0x00000002 [1] GPIO001 (0)
397 // 0x00000001 [0] GPIO000 (0)
398 io_ro_32 irqssummary_dormant_wake_nonsecure[2];
399
400 // (复制自数组索引0寄存器 IO_BANK0_INTR0 的描述，同样适用于其他数组索引)

401 _REG_(IO_BANK0_INTR0_OFFSET) // IO_BANK0_INTR0
402 // 原始中断
403 // 0x80000000 [31] GPIO07_EDGE_HIGH (0)
404 // 0x40000000 [30] GPIO07_EDGE_LOW (0)
405 // 0x20000000 [29] GPIO07_LEVEL_HIGH (0)
406 // 0x10000000 [28] GPIO07_LEVEL_LOW (0)

```

```

407 // 0x08000000 [27] GPIO06_EDGE_HIGH (0)
408 // 0x04000000 [26] GPIO06_EDGE_LOW (0)
409 // 0x02000000 [25] GPIO06_LEVEL_HIGH (0)
410 // 0x01000000 [24] GPIO06_LEVEL_LOW (0)
411 // 0x00800000 [23] GPIO05_EDGE_HIGH (0)
412 // 0x00400000 [22] GPIO05_EDGE_LOW (0)
413 // 0x00200000 [21] GPIO05_LEVEL_HIGH (0)
414 // 0x00100000 [20] GPIO05_LEVEL_LOW (0)
415 // 0x00080000 [19] GPIO04_EDGE_HIGH (0)
416 // 0x00040000 [18] GPIO04_EDGE_LOW (0)
417 // 0x00020000 [17] GPIO04_LEVEL_HIGH (0)
418 // 0x00010000 [16] GPIO04_LEVEL_LOW (0)
419 // 0x00008000 [15] GPIO03_EDGE_HIGH (0)
420 // 0x00004000 [14] GPIO03_EDGE_LOW (0)
421 // 0x00002000 [13] GPIO03_LEVEL_HIGH (0)
422 // 0x00001000 [12] GPIO03_LEVEL_LOW (0)
423 // 0x00000800 [11] GPIO02_EDGE_HIGH (0)
424 // 0x00000400 [10] GPIO02_EDGE_LOW (0)
425 // 0x00000200 [9] GPIO02_LEVEL_HIGH (0)
426 // 0x00000100 [8] GPIO02_LEVEL_LOW (0)
427 // 0x00000080 [7] GPIO01_EDGE_HIGH (0)
428 // 0x00000040 [6] GPIO01_EDGE_LOW (0)
429 // 0x00000020 [5] GPIO01_LEVEL_HIGH (0)
430 // 0x00000010 [4] GPIO01_LEVEL_LOW (0)
431 // 0x00000008 [3] GPIO00_EDGE_HIGH (0)
432 // 0x00000004 [2] GPIO00_EDGE_LOW (0)
433 // 0x00000002 [1] GPIO00_LEVEL_HIGH (0)
434 // 0x00000001 [0] GPIO00_LEVEL_LOW (0)
435 io_rw_32_intr[6];
436
437 union {
438 struct {
439 io_bank0_irq_ctrl_hw_t proc0_irq_ctrl;
440 io_bank0_irq_ctrl_hw_t proc1_irq_ctrl;
441 io_bank0_irq_ctrl_hw_t dormant_wake_irq_ctrl;
442 };
443 io_bank0_irq_ctrl_hw_t irq_ctrl[3];
444 };
445 } io_bank0_hw_t;

```

为 IO bank 1 的引脚控制寄存器定义了相似的结构体。默认情况下，所有引脚复位后均可直接使用，输入使能且输出禁用设置为0。无论如何，SDK中的`gpio_set_function`函数会设置输入使能，并清除输出禁止，从而激活引脚的IO缓冲区并将内部信号连接到外部。最后，将所需的功能选择写入IO控制寄存器（参见GPIO0\_CTRL以获取IO控制寄存器的示例）。

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_gpio/gpio.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/gpio.c) 第36至53行

```

36 // 选择此GPIO的功能，并确保引脚的输入和输出处于使能状态。
37 // 此外，还会清除输入、输出及中断覆盖位。
38 void gpio_set_function(uint gpio, gpio_function_t fn) {
39 check_gpio_param(gpio);
40 invalid_params_if(HARDWARE_GPIO, ((uint32_t)fn << IO_BANK0_GPIO00_CTRL_FUNCSEL_LSB) &
41 ~IO_BANK0_GPIO00_CTRL_FUNCSEL_BITS);
42 // 设置输入使能为开启，输出禁用为关闭
43 hw_write_masked(&pads_bank0_hw->io[gpio],
44 PADS_BANK0_GPIO00_IE_BITS,
45 PADS_BANK0_GPIO00_IE_BITS | PADS_BANK0_GPIO00_OD_BITS
46);
47 // 除 fsel 外，将所有字段清零；我们期望该 IO 按照外设的指示执行。
48 // 这不会影响诸如上拉/下拉，因为这些控制位于引脚控制寄存器中。
49 io_bank0_hw->io[gpio].ctrl = fn << IO_BANK0_GPIO00_CTRL_FUNCSEL_LSB;
50 // 当正确的外设接管引脚控制时，取消引脚隔离

```

```

50 hw_clear_bits(&pads_bank0_hw->io[gpio], PADS_BANK0_GPIO0_ISO_BITS);
51 }

```

### 9.10.2. 启用 GPIO 中断

SDK 提供了在 GPIO 引脚状态变化时触发中断的方法：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_gpio/gpio.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/gpio.c) 第186至196行

```

186 void gpio_set_irq_enabled(uint gpio, uint32_t events, bool enabled) {
187 // 此调用要么禁用中断，要么回调函数应已设置。
188 // 防止在未设置回调函数时启用中断。
189 assert(!enabled || irq_has_handler(IO_IRQ_BANK0));
190
191 // 每核心分离的掩码 / 强制 / 状态，故检查调用的核心，
192 // 并设置相应的 IRQ 控制。
193 io_bank0_irq_ctrl_hw_t *irq_ctrl_base = get_core_num() ?
194 &io_bank0_hw->proc1_irq_ctrl : &io_bank0_hw-
195 >proc0_irq_ctrl;
196 _gpio_set_irq_enabled(gpio, events, enabled, irq_ctrl_base);
196 }

```

`gpio_set_irq_enabled` 调用了更底层函数 `_gpio_set_irq_enabled`：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_gpio/gpio.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/gpio.c) Lines 173 - 184

```

173 static void _gpio_set_irq_enabled(uint gpio, uint32_t events, bool enabled,
174 io_bank0_irq_ctrl_hw_t *irq_ctrl_base) {
175 // 清除可能导致立即出现虚假中断处理的陈旧事件
176 gpio_acknowledge_irq(gpio, events);
177
178 io_rw_32 *en_reg = &irq_ctrl_base->inte[gpio / 8];
179 events <= 4 * (gpio % 8);
180
181 if (enabled)
182 hw_set_bits(en_reg, events);
183 else
184 hw_clear_bits(en_reg, events);
184 }

```

用户需提供一个回调函数指针，以在GPIO事件发生时被调用。使用该系统的示例应用程序为 `hello_gpio_irq`：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/gpio/hello\\_gpio\\_irq/hello\\_gpio\\_irq.c](https://github.com/raspberrypi/pico-examples/blob/master/gpio/hello_gpio_irq/hello_gpio_irq.c)

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符: BSD-3-Clause
5 */
6
7 #include <stdio.h>
8 #include "pico/stl.h"
9 #include "hardware/gpio.h"
10
11 #define GPIO_WATCH_PIN 2
12

```

```

13 static char event_str[128];
14
15 void gpio_event_string(char *buf, uint32_t events);
16
17 void gpio_callback(uint gpio, uint32_t events) {
18 // 将刚刚发生的 GPIO 事件写入 event_str
19 // 以便打印输出
20 gpio_event_string(event_str, events);
21 printf("GPIO %d %s\n", gpio, event_str);
22 }
23
24 int main() {
25 stdio_init_all();
26
27 printf("Hello GPIO IRQ\n");
28 gpio_init(GPIO_WATCH_PIN);
29 gpio_set_irq_enabled_with_callback(GPIO_WATCH_PIN, GPIO_IRQ_EDGE_RISE |
30 GPIO_IRQ_EDGE_FALL, true, &gpio_callback);
31
32 // 永久等待
33 while (1);
34 }
35
36 static const char *gpio_irq_str[] = {
37 "LEVEL_LOW", // 0x1
38 "LEVEL_HIGH", // 0x2
39 "EDGE_FALL", // 0x4
40 "EDGE_RISE" // 0x8
41 };
42
43 void gpio_event_string(char *buf, uint32_t events) {
44 for (uint i = 0; i < 4; i++) {
45 uint mask = (1 << i);
46 if (events & mask) {
47 // 将该事件字符串复制至用户缓冲区
48 const char *event_str = gpio_irq_str[i];
49 while (*event_str != '\0') {
50 *buf++ = *event_str++;
51 }
52 events &= ~mask;
53
54 // 如事件仍存在，追加 ", "
55 if (events) {
56 *buf++ = ',';
57 *buf++ = ' ';
58 }
59 }
60 }
61 *buf++ = '\0';
62 }

```

## 9.11. 寄存器列表

### 9.11.1. IO - 用户银行

用户银行 IO 寄存器起始于基地址 `0x40028000` (在 SDK 中定义为 `IO_BANK0_BASE`)。

表649 IO\_BANK0  
寄存器列表

偏移量	名称	说明
0x000	GPIO0_STATUS	
0x004	GPIO0_CTRL	
0x008	GPIO1_STATUS	
0x00c	GPIO1_CTRL	
0x010	GPIO2_STATUS	
0x014	GPIO2_CTRL	
0x018	GPIO3_STATUS	
0x01c	GPIO3_CTRL	
0x020	GPIO4_STATUS	
0x024	GPIO4_CTRL	
0x028	GPIO5_STATUS	
0x02c	GPIO5_CTRL	
0x030	GPIO6_STATUS	
0x034	GPIO6_CTRL	
0x038	GPIO7_STATUS	
0x03c	GPIO7_CTRL	
0x040	GPIO8_STATUS	
0x044	GPIO8_CTRL	
0x048	GPIO9_STATUS	
0x04c	GPIO9_CTRL	
0x050	GPIO10_STATUS	
0x054	GPIO10_CTRL	
0x058	GPIO11_STATUS	
0x05c	GPIO11_CTRL	
0x060	GPIO12_STATUS	
0x064	GPIO12_CTRL	
0x068	GPIO13_STATUS	
0x06c	GPIO13_CTRL	
0x070	GPIO14_STATUS	
0x074	GPIO14_CTRL	
0x078	GPIO15_STATUS	
0x07c	GPIO15_CTRL	
0x080	GPIO16_STATUS	
0x084	GPIO16_CTRL	
0x088	GPIO17_STATUS	
0x08c	GPIO17_CTRL	

偏移量	名称	说明
0x090	GPIO18_STATUS	
0x094	GPIO18_CTRL	
0x098	GPIO19_STATUS	
0x09c	GPIO19_CTRL	
0x0a0	GPIO20_STATUS	
0x0a4	GPIO20_CTRL	
0x0a8	GPIO21_STATUS	
0x0ac	GPIO21_CTRL	
0x0b0	GPIO22_STATUS	
0x0b4	GPIO22_CTRL	
0x0b8	GPIO23_STATUS	
0x0bc	GPIO23_CTRL	
0x0c0	GPIO24_STATUS	
0x0c4	GPIO24_CTRL	
0x0c8	GPIO25_STATUS	
0x0cc	GPIO25_CTRL	
0x0d0	GPIO26_STATUS	
0x0d4	GPIO26_CTRL	
0x0d8	GPIO27_STATUS	
0x0dc	GPIO27_CTRL	
0x0e0	GPIO28_STATUS	
0x0e4	GPIO28_CTRL	
0x0e8	GPIO29_STATUS	
0x0ec	GPIO29_CTRL	
0x0f0	GPIO30_STATUS	
0x0f4	GPIO30_CTRL	
0x0f8	GPIO31_STATUS	
0xfc	GPIO31_CTRL	
0x100	GPIO32_STATUS	
0x104	GPIO32_CTRL	
0x108	GPIO33_STATUS	
0x10c	GPIO33_CTRL	
0x110	GPIO34_STATUS	
0x114	GPIO34_CTRL	
0x118	GPIO35_STATUS	
0x11c	GPIO35_CTRL	

偏移量	名称	说明
0x120	GPIO36_STATUS	
0x124	GPIO36_CTRL	
0x128	GPIO37_STATUS	
0x12c	GPIO37_CTRL	
0x130	GPIO38_STATUS	
0x134	GPIO38_CTRL	
0x138	GPIO39_STATUS	
0x13c	GPIO39_CTRL	
0x140	GPIO40_STATUS	
0x144	GPIO40_CTRL	
0x148	GPIO41_STATUS	
0x14c	GPIO41_CTRL	
0x150	GPIO42_STATUS	
0x154	GPIO42_CTRL	
0x158	GPIO43_STATUS	
0x15c	GPIO43_CTRL	
0x160	GPIO44_STATUS	
0x164	GPIO44_CTRL	
0x168	GPIO45_STATUS	
0x16c	GPIO45_CTRL	
0x170	GPIO46_STATUS	
0x174	GPIO46_CTRL	
0x178	GPIO47_STATUS	
0x17c	GPIO47_CTRL	
0x200	IRQSUMMARY_PROC0_SECURE0	
0x204	IRQSUMMARY_PROC0_SECURE1	
0x208	IRQSUMMARY_PROC0_NONSECURE0	
0x20c	IRQSUMMARY_PROC0_NONSECURE1	
0x210	IRQSUMMARY_PROC1_SECURE0	
0x214	IRQSUMMARY_PROC1_SECURE1	
0x218	IRQSUMMARY_PROC1_NONSECURE0	
0x21c	IRQSUMMARY_PROC1_NONSECURE1	
0x220	IRQSUMMARY_COMA_WAKE_SECURE 0	
0x224	IRQSUMMARY_COMA_WAKE_SECURE 1	

偏移量	名称	说明
0x228	IRQSUMMARY_COMA_WAKE_NONSE_CURE0	
0x22c	IRQSUMMARY_COMA_WAKE_NONSE_CURE1	
0x230	INTR0	原始中断
0x234	INTR1	原始中断
0x238	INTR2	原始中断
0x23c	INTR3	原始中断
0x240	INTR4	原始中断
0x244	INTR5	原始中断
0x248	PROC0_INTE0	proc0的中断使能
0x24c	PROC0_INTE1	proc0的中断使能
0x250	PROC0_INTE2	proc0的中断使能
0x254	PROC0_INTE3	proc0的中断使能
0x258	PROC0_INTE4	proc0的中断使能
0x25c	PROC0_INTE5	proc0的中断使能
0x260	PROC0_INTFO	proc0的中断强制
0x264	PROC0_INTF1	proc0的中断强制
0x268	PROC0_INTF2	proc0的中断强制
0x26c	PROC0_INTF3	proc0的中断强制
0x270	PROC0_INTF4	proc0的中断强制
0x274	PROC0_INTF5	proc0的中断强制
0x278	PROC0_INTS0	proc0 屏蔽及强制后的中断状态
0x27c	PROC0_INTS1	proc0 屏蔽及强制后的中断状态
0x280	PROC0_INTS2	proc0 屏蔽及强制后的中断状态
0x284	PROC0_INTS3	proc0 屏蔽及强制后的中断状态
0x288	PROC0_INTS4	proc0 屏蔽及强制后的中断状态
0x28c	PROC0_INTS5	proc0 屏蔽及强制后的中断状态
0x290	PROC1_INTE0	proc1 中断使能
0x294	PROC1_INTE1	proc1 中断使能
0x298	PROC1_INTE2	proc1 中断使能
0x29c	PROC1_INTE3	proc1 中断使能
0x2a0	PROC1_INTE4	proc1 中断使能
0x2a4	PROC1_INTE5	proc1 中断使能
0x2a8	PROC1_INTFO	proc1 中断强制
0x2ac	PROC1_INTF1	proc1 中断强制

偏移量	名称	说明
0x2b0	PROC1_INTF2	proc1 中断强制
0x2b4	PROC1_INTF3	proc1 中断强制
0x2b8	PROC1_INTF4	proc1 中断强制
0x2bc	PROC1_INTF5	proc1 中断强制
0x2c0	PROC1_INTS0	proc1 掩码与强制后的中断状态
0x2c4	PROC1_INTS1	proc1 掩码与强制后的中断状态
0x2c8	PROC1_INTS2	proc1 掩码与强制后的中断状态
0x2cc	PROC1_INTS3	proc1 掩码与强制后的中断状态
0x2d0	PROC1_INTS4	proc1 掩码与强制后的中断状态
0x2d4	PROC1_INTS5	proc1 掩码与强制后的中断状态
0x2d8	DORMANT_WAKE_INTE0	dormant_wake 的中断使能
0x2dc	DORMANT_WAKE_INTE1	dormant_wake 的中断使能
0x2e0	DORMANT_WAKE_INTE2	dormant_wake 的中断使能
0x2e4	DORMANT_WAKE_INTE3	dormant_wake 的中断使能
0x2e8	DORMANT_WAKE_INTE4	dormant_wake 的中断使能
0x2ec	DORMANT_WAKE_INTE5	dormant_wake 的中断使能
0x2f0	DORMANT_WAKE_INTFO	dormant_wake 的中断强制
0x2f4	DORMANT_WAKE_INTF1	dormant_wake 的中断强制
0x2f8	DORMANT_WAKE_INTF2	dormant_wake 的中断强制
0x2fc	DORMANT_WAKE_INTF3	dormant_wake 的中断强制
0x300	DORMANT_WAKE_INTF4	dormant_wake 的中断强制
0x304	DORMANT_WAKE_INTF5	dormant_wake 的中断强制
0x308	DORMANT_WAKE_INTSO	dormant_wake 掩码与强制后的中断状态
0x30c	DORMANT_WAKE_INTS1	dormant_wake 掩码与强制后的中断状态
0x310	DORMANT_WAKE_INTS2	dormant_wake 掩码与强制后的中断状态
0x314	DORMANT_WAKE_INTS3	dormant_wake 掩码与强制后的中断状态
0x318	DORMANT_WAKE_INTS4	dormant_wake 掩码与强制后的中断状态
0x31c	DORMANT_WAKE_INTS5	dormant_wake 掩码与强制后的中断状态

## IO\_BANK0: GPIO0\_STATUS 寄存器

偏移：0x000

表 650。  
GPIO0\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0

位	描述	类型	复位
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO0\_CTRL 寄存器

偏移量: 0x004

表 651。  
GPIO0\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		

位	描述	类型	复位
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL:</b> 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x00 → JTAG_TCK		
	0x01 → SPI0_RX		
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_0		
	0x05 → SIO_0		
	0x06 → PIO0_0		
	0x07 → PIO1_0		
	0x08 → PIO2_0		
	0x09 → XIP_SS_N_1		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO1\_STATUS 寄存器

偏移: 0x008

表 652。  
GPIO1\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC:</b> 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD:</b> 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD:</b> 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD:</b> 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO1\_CTRL 寄存器

偏移量: 0x00c

表 653。  
GPIO1\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-

位	描述	类型	复位
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → JTAG_TMS		
	0x01 → SPI0_SS_N		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_0		

位	描述	类型	复位
	0x05 → SIO_1		
	0x06 → PIO0_1		
	0x07 → PIO1_1		
	0x08 → PIO2_1		
	0x09 → CORESIGHT_TRACECLK		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO2\_STATUS 寄存器

偏移量: 0x010

表654。  
GPIO2\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO2\_CTRL 寄存器

偏移: 0x014

表655。  
GPIO2\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		

位	描述	类型	复位
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x00 → JTAG_TDI		
	0x01 → SPI0_SCLK		
	0x02 → UART0_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_1		
	0x05 → SIO_2		
	0x06 → PIO0_2		
	0x07 → PIO1_2		
	0x08 → PIO2_2		
	0x09 → CORESIGHT_TRACEDEDATA_0		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART0_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO3\_STATUS 寄存器

偏移量: 0x018

表 656。  
GPIO3\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO3\_CTRL 寄存器

偏移量: 0x01c

表 657。  
GPIO3\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL:</b> 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → JTAG_TDO		
	0x01 → SPI0_TX		
	0x02 → UART0_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_1		
	0x05 → SIO_3		
	0x06 → PIO0_3		
	0x07 → PIO1_3		
	0x08 → PIO2_3		
	0x09 → CORESIGHT_TRACEDEATA_1		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO4\_STATUS 寄存器

偏移: 0x020

表 658  
GPIO4\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC:</b> 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD:</b> 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD:</b> 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD:</b> 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO4\_CTRL 寄存器

偏移: 0x024

表 659  
GPIO4\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x01 → SPI0_RX		
	0x02 → UART1_TX		

位	描述	类型	复位
	0x03 → I2C0_SDA		
	0x04 → PWM_A_2		
	0x05 → SIO_4		
	0x06 → PIO0_4		
	0x07 → PIO1_4		
	0x08 → PIO2_4		
	0x09 → CORESIGHT_TRACE DATA_2		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO5\_STATUS 寄存器

偏移: 0x028

表 660  
GPIO5\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO5\_CTRL 寄存器

偏移量: 0x02c

表 661  
GPIO5\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SS_N		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_2		
	0x05 → SIO_5		
	0x06 → PIO0_5		
	0x07 → PIO1_5		
	0x08 → PIO2_5		
	0x09 → CORESIGHT_TRACE DATA_3		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO6\_STATUS 寄存器

偏移量: 0x030

表662  
GPIO6\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO6\_CTRL寄存器

偏移: 0x034

表663  
GPIO6\_CTRL寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL:</b> 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_3		
	0x05 → SIO_6		
	0x06 → PIO0_6		
	0x07 → PIO1_6		
	0x08 → PIO2_6		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO7\_STATUS 寄存器

偏移量: 0x038

表 664  
GPIO7\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC:</b> 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD:</b> 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD:</b> 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD:</b> 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO7\_CTRL 寄存器

偏移量: 0x03c

表 665  
GPIO7\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_TX		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_3		

位	描述	类型	复位
	0x05 → SIO_7		
	0x06 → PIO0_7		
	0x07 → PIO1_7		
	0x08 → PIO2_7		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO8\_STATUS 寄存器

偏移量: 0x040

表 666。  
GPIO8\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO8\_CTRL 寄存器

偏移量: 0x044

表 667。  
GPIO8\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		

位	描述	类型	复位
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x01 → SPI1_RX		
	0x02 → UART1_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_4		
	0x05 → SIO_8		
	0x06 → PIO0_8		
	0x07 → PIO1_8		
	0x08 → PIO2_8		
	0x09 → XIP_SS_N_1		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO9\_STATUS 寄存器

偏移量: 0x048

表 668  
GPIO9\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-

位	描述	类型	复位
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO9\_CTRL 寄存器

偏移量: 0x04c

表 669  
GPIO9\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SS_N		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_4		
	0x05 → SIO_9		
	0x06 → PIO0_9		
	0x07 → PIO1_9		
	0x08 → PIO2_9		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO10\_STATUS 寄存器

偏移量: 0x050

表 670  
GPIO10\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO10\_CTRL 寄存器

偏移: 0x054

表 671。  
GPIO10\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_5		

位	描述	类型	复位
	0x05 → SIO_10		
	0x06 → PIO0_10		
	0x07 → PIO1_10		
	0x08 → PIO2_10		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO11\_STATUS 寄存器

偏移: 0x058

表 672。  
GPIO11\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO11\_CTRL 寄存器

偏移: 0x05c

表 673。  
GPIO11\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		

位	描述	类型	复位
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_TX		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_5		
	0x05 → SIO_11		
	0x06 → PIO0_11		
	0x07 → PIO1_11		
	0x08 → PIO2_11		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO12\_STATUS 寄存器

偏移量: 0x060

表 674。  
GPIO12\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-

位	描述	类型	复位
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO12\_CTRL 寄存器

偏移量: 0x064

表 675。  
GPIO12\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → HSTX_0		
	0x01 → SPI1_RX		
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_6		
	0x05 → SIO_12		
	0x06 → PIO0_12		
	0x07 → PIO1_12		
	0x08 → PIO2_12		
	0x09 → CLOCKS_GPIN_0		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO13\_STATUS 寄存器

偏移: 0x068

表 676  
GPIO13\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO13\_CTRL 寄存器

偏移: 0x06c

表 677  
GPIO13\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → HSTX_1		
	0x01 → SPI1_SS_N		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		

位	描述	类型	复位
	0x04 → PWM_B_6		
	0x05 → SIO_13		
	0x06 → PIO0_13		
	0x07 → PIO1_13		
	0x08 → PIO2_13		
	0x09 → CLOCKS_GPOUT_0		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO14\_STATUS 寄存器

偏移: 0x070

表 678  
GPIO14\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO14\_CTRL 寄存器

偏移量: 0x074

表 679  
GPIO14\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		

位	描述	类型	复位
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → HSTX_2		
	0x01 → SPI1_SCLK		
	0x02 → UART0_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_7		
	0x05 → SIO_14		
	0x06 → PIO0_14		
	0x07 → PIO1_14		
	0x08 → PIO2_14		
	0x09 → CLOCKS_GPIN_1		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART0_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO15\_STATUS 寄存器

偏移量: 0x078

表 680  
GPIO15\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO15\_CTRL 寄存器

偏移量: 0x07c

表 681。  
GPIO15\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL:</b> 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → HSTX_3		
	0x01 → SPI1_TX		
	0x02 → UART0_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_7		
	0x05 → SIO_15		
	0x06 → PIO0_15		
	0x07 → PIO1_15		
	0x08 → PIO2_15		
	0x09 → CLOCKS_GPOUT_1		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO16\_STATUS 寄存器

偏移: 0x080

表 682。  
GPIO16\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC:</b> 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD:</b> 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD:</b> 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD:</b> 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO16\_CTRL 寄存器

偏移: 0x084

表 683.  
GPIO16\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x00 → HSTX_4		
	0x01 → SPI0_RX		

位	描述	类型	复位
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_0		
	0x05 → SIO_16		
	0x06 → PIO0_16		
	0x07 → PIO1_16		
	0x08 → PIO2_16		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO17\_STATUS 寄存器

偏移: 0x088

表684  
GPIO17\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO17\_CTRL 寄存器

偏移: 0x08c

表685  
GPIO17\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → HSTX_5		
	0x01 → SPI0_SS_N		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_0		
	0x05 → SIO_17		
	0x06 → PIO0_17		
	0x07 → PIO1_17		
	0x08 → PIO2_17		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO18\_STATUS 寄存器

偏移: 0x090

表686  
GPIO18\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO18\_CTRL 寄存器

偏移: 0x094

表687  
GPIO18\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL:</b> 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → HSTX_6		
	0x01 → SPI0_SCLK		
	0x02 → UART0_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_1		
	0x05 → SIO_18		
	0x06 → PIO0_18		
	0x07 → PIO1_18		
	0x08 → PIO2_18		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART0_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO19\_STATUS 寄存器

偏移: 0x098

表688  
GPIO19\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC:</b> 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD:</b> 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD:</b> 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD:</b> 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO19\_CTRL 寄存器

偏移: 0x09c

表689  
GPIO19\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x00 → HSTX_7		
	0x01 → SPI0_TX		
	0x02 → UART0_RTS		

位	描述	类型	复位
	0x03 → I2C1_SCL		
	0x04 → PWM_B_1		
	0x05 → SIO_19		
	0x06 → PIO0_19		
	0x07 → PIO1_19		
	0x08 → PIO2_19		
	0x09 → XIP_SS_N_1		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_BANK0：GPIO20\_STATUS 寄存器

偏移：0x0a0

表690  
GPIO20\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> ：覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> ：滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> ：寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> ：寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0：GPIO20\_CTRL 寄存器

偏移：0x0a4

表691  
GPIO20\_CTRL寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_RX		
	0x02 → UART1_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_2		
	0x05 → SIO_20		
	0x06 → PIO0_20		
	0x07 → PIO1_20		
	0x08 → PIO2_20		
	0x09 → CLOCKS_GPIN_0		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO21\_STATUS 寄存器

偏移：0x0a8

表692  
GPIO21\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO21\_CTRL寄存器

偏移: 0x0ac

表693  
GPIO21\_CTRL寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL:</b> 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SS_N		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_2		
	0x05 → SIO_21		
	0x06 → PIO0_21		
	0x07 → PIO1_21		
	0x08 → PIO2_21		
	0x09 → CLOCKS_GPOUT_0		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO22\_STATUS 寄存器

偏移量: 0x0b0

表694。  
GPIO22\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC:</b> 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD:</b> 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD:</b> 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD:</b> 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO22\_CTRL 寄存器

偏移：0x0b4

表695。  
GPIO22\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_3		

位	描述	类型	复位
	0x05 → SIO_22		
	0x06 → PIO0_22		
	0x07 → PIO1_22		
	0x08 → PIO2_22		
	0x09 → CLOCKS_GPIN_1		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO23\_STATUS 寄存器

偏移: 0x0b8

表696。  
GPIO23\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO23\_CTRL 寄存器

偏移: 0x0bc

表697。  
GPIO23\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		

位	描述	类型	复位
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_TX		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_3		
	0x05 → SIO_23		
	0x06 → PIO0_23		
	0x07 → PIO1_23		
	0x08 → PIO2_23		
	0x09 → CLOCKS_GPOUT_1		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO24\_STATUS 寄存器

偏移: 0x0c0

表 698  
GPIO24\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO24\_CTRL 寄存器

偏移: 0x0c4

表 699  
GPIO24\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_RX		
	0x02 → UART1_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_4		
	0x05 → SIO_24		
	0x06 → PIO0_24		
	0x07 → PIO1_24		
	0x08 → PIO2_24		
	0x09 → CLOCKS_GPOUT_2		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO25\_STATUS 寄存器

偏移: 0x0c8

表 700  
GPIO25\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO25\_CTRL 寄存器

偏移: 0x0cc

表 701。  
GPIO25\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SS_N		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_4		

位	描述	类型	复位
	0x05 → SIO_25		
	0x06 → PIO0_25		
	0x07 → PIO1_25		
	0x08 → PIO2_25		
	0x09 → CLOCKS_GPOUT_3		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO26\_STATUS 寄存器

偏移: 0x0d0

表 702。  
GPIO26\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO26\_CTRL 寄存器

偏移: 0x0d4

表 703。  
GPIO26\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		

位	描述	类型	复位
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x01 → SPI1_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_5		
	0x05 → SIO_26		
	0x06 → PIO0_26		
	0x07 → PIO1_26		
	0x08 → PIO2_26		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO27\_STATUS 寄存器

偏移: 0x0d8

表 704  
GPIO27\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-

位	描述	类型	复位
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO27\_CTRL 寄存器

偏移: 0x0dc

表 705  
GPIO27\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_TX		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_5		
	0x05 → SIO_27		
	0x06 → PIO0_27		
	0x07 → PIO1_27		
	0x08 → PIO2_27		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO28\_STATUS 寄存器

偏移: 0x0e0

表 706  
GPIO28\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO28\_CTRL 寄存器

偏移: 0x0e4

表 707  
GPIO28\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_RX		
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_6		

位	描述	类型	复位
	0x05 → SIO_28		
	0x06 → PIO0_28		
	0x07 → PIO1_28		
	0x08 → PIO2_28		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0：GPIO29\_STATUS 寄存器

偏移: 0x0e8

表 708  
GPIO29\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0：GPIO29\_CTRL 寄存器

偏移量: 0x0ec

表 709  
GPIO29\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		

位	描述	类型	复位
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SS_N		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_6		
	0x05 → SIO_29		
	0x06 → PIO0_29		
	0x07 → PIO1_29		
	0x08 → PIO2_29		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO30\_STATUS 寄存器

偏移量: 0x0f0

表 710  
GPIO30\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-

位	描述	类型	复位
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO30\_CTRL 寄存器

偏移量: 0x0f4

表711  
GPIO30\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		

位	描述	类型	复位
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SCLK		
	0x02 → UART0_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_7		
	0x05 → SIO_30		
	0x06 → PIO0_30		
	0x07 → PIO1_30		
	0x08 → PIO2_30		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART0_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO31\_STATUS 寄存器

偏移量: 0x0f8

表712  
GPIO31\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO31\_CTRL 寄存器

偏移量: 0x0fc

表713  
GPIO31\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_TX		
	0x02 → UART0_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_7		

位	描述	类型	复位
	0x05 → SIO_31		
	0x06 → PIO0_31		
	0x07 → PIO1_31		
	0x08 → PIO2_31		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO32\_STATUS 寄存器

偏移量: 0x100

表 714。  
GPIO32\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO32\_CTRL 寄存器

偏移量: 0x104

表 715。  
GPIO32\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		

位	描述	类型	复位
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x01 → SPI0_RX		
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_8		
	0x05 → SIO_32		
	0x06 → PIO0_32		
	0x07 → PIO1_32		
	0x08 → PIO2_32		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO33\_STATUS 寄存器

偏移量: 0x108

表 716。  
GPIO33\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0

位	描述	类型	复位
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO33\_CTRL 寄存器

偏移量: 0x10c

表 717.  
GPIO33\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		

位	描述	类型	复位
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SS_N		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_8		
	0x05 → SIO_33		
	0x06 → PIO0_33		
	0x07 → PIO1_33		
	0x08 → PIO2_33		
	0xa → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO34\_STATUS 寄存器

偏移量: 0x110

表 718  
GPIO34\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO34\_CTRL 寄存器

偏移量: 0x114

表 719  
GPIO34\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-

位	描述	类型	复位
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SCLK		
	0x02 → UART0_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_9		
	0x05 → SIO_34		

位	描述	类型	复位
	0x06 → PIO0_34		
	0x07 → PIO1_34		
	0x08 → PIO2_34		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART0_TX		
	0x1f → NULL		

## IO\_BANK0：GPIO35\_STATUS 寄存器

偏移: 0x118

表 720  
GPIO35\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0：GPIO35\_CTRL 寄存器

偏移: 0x11c

表 721  
GPIO35\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		

位	描述	类型	复位
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_TX		
	0x02 → UART0_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_9		
	0x05 → SIO_35		
	0x06 → PIO0_35		
	0x07 → PIO1_35		
	0x08 → PIO2_35		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO36\_STATUS 寄存器

偏移: 0x120

表 722  
GPIO36\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0

位	描述	类型	复位
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO36\_CTRL 寄存器

偏移: 0x124

表723。  
GPIO36\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		

位	描述	类型	复位
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_RX		
	0x02 → UART1_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_10		
	0x05 → SIO_36		
	0x06 → PIO0_36		
	0x07 → PIO1_36		
	0x08 → PIO2_36		
	0xa → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO37\_STATUS 寄存器

偏移：0x128

表724。  
GPIO37\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO37\_CTRL 寄存器

偏移：0x12c

表725。  
GPIO37\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-

位	描述	类型	复位
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SS_N		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_10		
	0x05 → SIO_37		

位	描述	类型	复位
	0x06 → PIO0_37		
	0x07 → PIO1_37		
	0x08 → PIO2_37		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO38\_STATUS 寄存器

偏移量: 0x130

表 726。  
GPIO38\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO38\_CTRL 寄存器

偏移量: 0x134

表 727。  
GPIO38\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		

位	描述	类型	复位
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_11		
	0x05 → SIO_38		
	0x06 → PIO0_38		
	0x07 → PIO1_38		
	0x08 → PIO2_38		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO39\_STATUS 寄存器

偏移: 0x138

表728。  
GPIO39\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-

位	描述	类型	复位
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO39\_CTRL寄存器

偏移: 0x13C

表729。  
GPIO39\_CTRL寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		

位	描述	类型	复位
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI0_TX		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_11		
	0x05 → SIO_39		
	0x06 → PIO0_39		
	0x07 → PIO1_39		
	0x08 → PIO2_39		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO40\_STATUS寄存器

偏移: 0x140

表730。  
GPIO40\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO40\_CTRL寄存器

偏移: 0x144

表731  
GPIO40\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_RX		
	0x02 → UART1_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_8		

位	描述	类型	复位
	0x05 → SIO_40		
	0x06 → PIO0_40		
	0x07 → PIO1_40		
	0x08 → PIO2_40		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x1f → NULL		

## IO\_BANK0：GPIO41\_STATUS 寄存器

偏移：0x148

表732  
GPIO41\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> ：覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> ：滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> ：寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> ：寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0：GPIO41\_CTRL 寄存器

偏移：0x14c

表733  
GPIO41\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		

位	描述	类型	复位
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SS_N		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_8		
	0x05 → SIO_41		
	0x06 → PIO0_41		
	0x07 → PIO1_41		
	0x08 → PIO2_41		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO42\_STATUS 寄存器

偏移: 0x150

表 734  
GPIO42\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-

位	描述	类型	复位
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO42\_CTRL 寄存器

偏移: 0x154

表 735  
GPIO42\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		

位	描述	类型	复位
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_9		
	0x05 → SIO_42		
	0x06 → PIO0_42		
	0x07 → PIO1_42		
	0x08 → PIO2_42		
	0x0a → USB_MUXING_OVERCURR_DETECT		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_BANK0: GPIO43\_STATUS 寄存器

偏移: 0x158

表 736  
GPIO43\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO43\_CTRL 寄存器

偏移: 0x15c

表737  
GPIO43\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_TX		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_9		

位	描述	类型	复位
	0x05 → SIO_43		
	0x06 → PIO0_43		
	0x07 → PIO1_43		
	0x08 → PIO2_43		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_BANK0: GPIO44\_STATUS 寄存器

偏移: 0x160

表738  
GPIO44\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO44\_CTRL 寄存器

偏移: 0x164

表739  
GPIO44\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		

位	描述	类型	复位
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT: 驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x01 → SPI1_RX		
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x04 → PWM_A_10		
	0x05 → SIO_44		
	0x06 → PIO0_44		
	0x07 → PIO1_44		
	0x08 → PIO2_44		
	0x0a → USB_MUXING_VBUS_EN		
	0x1f → NULL		

## IO\_BANK0: GPIO45\_STATUS 寄存器

偏移: 0x168

表 740  
GPIO45\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0

位	描述	类型	复位
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO45\_CTRL 寄存器

偏移: 0x16c

表 741  
GPIO45\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由 funcsel 选择的外设信号		

位	描述	类型	复位
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SS_N		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		
	0x04 → PWM_B_10		
	0x05 → SIO_45		
	0x06 → PIO0_45		
	0x07 → PIO1_45		
	0x08 → PIO2_45		
	0xa → USB_MUXING_OVERCURR_DETECT		
	0x1f → NULL		

## IO\_BANK0: GPIO46\_STATUS 寄存器

偏移: 0x170

表 742  
GPIO46\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0: GPIO46\_CTRL 寄存器

偏移: 0x174

表 743  
GPIO46\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-

位	描述	类型	复位
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_SCLK		
	0x02 → UART0_CTS		
	0x03 → I2C1_SDA		
	0x04 → PWM_A_11		
	0x05 → SIO_46		

位	描述	类型	复位
	0x06 → PIO0_46		
	0x07 → PIO1_46		
	0x08 → PIO2_46		
	0x0a → USB_MUXING_VBUS_DETECT		
	0x0b → UART0_TX		
	0x1f → NULL		

## IO\_BANK0：GPIO47\_STATUS 寄存器

偏移：0x178

表 744  
GPIO47\_STATUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> ：覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> ：滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> ：寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> ：寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_BANK0：GPIO47\_CTRL 寄存器

偏移：0x17c

表 745  
GPIO47\_CTRL 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		

位	描述	类型	复位
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x01 → SPI1_TX		
	0x02 → UART0_RTS		
	0x03 → I2C1_SCL		
	0x04 → PWM_B_11		
	0x05 → SIO_47		
	0x06 → PIO0_47		
	0x07 → PIO1_47		
	0x08 → PIO2_47		
	0x09 → XIP_SS_N_1		
	0x0a → USB_MUXING_VBUS_EN		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_BANK0: IRQSUMMARY\_PROCO\_SECURE0 寄存器

偏移: 0x200

表746。  
IRQSUMMARY\_PROCO\_SECURE0 寄存器

位	描述	类型	复位
31	<b>GPIO31</b>	只读	0x0

位	描述	类型	复位
30	<b>GPIO30</b>	只读	0x0
29	<b>GPIO29</b>	只读	0x0
28	<b>GPIO28</b>	只读	0x0
27	<b>GPIO27</b>	只读	0x0
26	<b>GPIO26</b>	只读	0x0
25	<b>GPIO25</b>	只读	0x0
24	<b>GPIO24</b>	只读	0x0
23	<b>GPIO23</b>	只读	0x0
22	<b>GPIO22</b>	只读	0x0
21	<b>GPIO21</b>	只读	0x0
20	<b>GPIO20</b>	只读	0x0
19	<b>GPIO19</b>	只读	0x0
18	<b>GPIO18</b>	只读	0x0
17	<b>GPIO17</b>	只读	0x0
16	<b>GPIO16</b>	只读	0x0
15	<b>GPIO15</b>	只读	0x0
14	<b>GPIO14</b>	只读	0x0
13	<b>GPIO13</b>	只读	0x0
12	<b>GPIO12</b>	只读	0x0
11	<b>GPIO11</b>	只读	0x0
10	<b>GPIO10</b>	只读	0x0
9	<b>GPIO9</b>	只读	0x0
8	<b>GPIO8</b>	只读	0x0
7	<b>GPIO7</b>	只读	0x0
6	<b>GPIO6</b>	只读	0x0
5	<b>GPIO5</b>	只读	0x0
4	<b>GPIO4</b>	只读	0x0
3	<b>GPIO3</b>	只读	0x0
2	<b>GPIO2</b>	只读	0x0
1	<b>GPIO1</b>	只读	0x0
0	<b>GPIO0</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROC0\_SECURE1 寄存器

偏移: 0x204

表747。  
IRQSUMMARY\_PROC0\_SECURE1 寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15	<b>GPIO47</b>	只读	0x0
14	<b>GPIO46</b>	只读	0x0
13	<b>GPIO45</b>	只读	0x0
12	<b>GPIO44</b>	只读	0x0
11	<b>GPIO43</b>	只读	0x0
10	<b>GPIO42</b>	只读	0x0
9	<b>GPIO41</b>	只读	0x0
8	<b>GPIO40</b>	只读	0x0
7	<b>GPIO39</b>	只读	0x0
6	<b>GPIO38</b>	只读	0x0
5	<b>GPIO37</b>	只读	0x0
4	<b>GPIO36</b>	只读	0x0
3	<b>GPIO35</b>	只读	0x0
2	<b>GPIO34</b>	只读	0x0
1	<b>GPIO33</b>	只读	0x0
0	<b>GPIO32</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROCO\_NONSECURE0 寄存器

偏移: 0x208

表748.  
IRQSUMMARY\_PROCO  
\_NONSECURE0  
寄存器

位	描述	类型	复位
31	<b>GPIO31</b>	只读	0x0
30	<b>GPIO30</b>	只读	0x0
29	<b>GPIO29</b>	只读	0x0
28	<b>GPIO28</b>	只读	0x0
27	<b>GPIO27</b>	只读	0x0
26	<b>GPIO26</b>	只读	0x0
25	<b>GPIO25</b>	只读	0x0
24	<b>GPIO24</b>	只读	0x0
23	<b>GPIO23</b>	只读	0x0
22	<b>GPIO22</b>	只读	0x0
21	<b>GPIO21</b>	只读	0x0
20	<b>GPIO20</b>	只读	0x0
19	<b>GPIO19</b>	只读	0x0
18	<b>GPIO18</b>	只读	0x0
17	<b>GPIO17</b>	只读	0x0
16	<b>GPIO16</b>	只读	0x0

位	描述	类型	复位
15	<b>GPIO15</b>	只读	0x0
14	<b>GPIO14</b>	只读	0x0
13	<b>GPIO13</b>	只读	0x0
12	<b>GPIO12</b>	只读	0x0
11	<b>GPIO11</b>	只读	0x0
10	<b>GPIO10</b>	只读	0x0
9	<b>GPIO9</b>	只读	0x0
8	<b>GPIO8</b>	只读	0x0
7	<b>GPIO7</b>	只读	0x0
6	<b>GPIO6</b>	只读	0x0
5	<b>GPIO5</b>	只读	0x0
4	<b>GPIO4</b>	只读	0x0
3	<b>GPIO3</b>	只读	0x0
2	<b>GPIO2</b>	只读	0x0
1	<b>GPIO1</b>	只读	0x0
0	<b>GPIO0</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROC0\_NONSECURE1寄存器

偏移: 0x20c

表749.  
IRQSUMMARY\_PROC0  
\_NONSECURE1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>GPIO47</b>	只读	0x0
14	<b>GPIO46</b>	只读	0x0
13	<b>GPIO45</b>	只读	0x0
12	<b>GPIO44</b>	只读	0x0
11	<b>GPIO43</b>	只读	0x0
10	<b>GPIO42</b>	只读	0x0
9	<b>GPIO41</b>	只读	0x0
8	<b>GPIO40</b>	只读	0x0
7	<b>GPIO39</b>	只读	0x0
6	<b>GPIO38</b>	只读	0x0
5	<b>GPIO37</b>	只读	0x0
4	<b>GPIO36</b>	只读	0x0
3	<b>GPIO35</b>	只读	0x0
2	<b>GPIO34</b>	只读	0x0
1	<b>GPIO33</b>	只读	0x0

位	描述	类型	复位
0	<b>GPIO32</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROC1\_SECURE0寄存器

偏移: 0x210

表750.  
IRQSUMMARY\_PROC1  
\_SECURE0寄存器

位	描述	类型	复位
31	<b>GPIO31</b>	只读	0x0
30	<b>GPIO30</b>	只读	0x0
29	<b>GPIO29</b>	只读	0x0
28	<b>GPIO28</b>	只读	0x0
27	<b>GPIO27</b>	只读	0x0
26	<b>GPIO26</b>	只读	0x0
25	<b>GPIO25</b>	只读	0x0
24	<b>GPIO24</b>	只读	0x0
23	<b>GPIO23</b>	只读	0x0
22	<b>GPIO22</b>	只读	0x0
21	<b>GPIO21</b>	只读	0x0
20	<b>GPIO20</b>	只读	0x0
19	<b>GPIO19</b>	只读	0x0
18	<b>GPIO18</b>	只读	0x0
17	<b>GPIO17</b>	只读	0x0
16	<b>GPIO16</b>	只读	0x0
15	<b>GPIO15</b>	只读	0x0
14	<b>GPIO14</b>	只读	0x0
13	<b>GPIO13</b>	只读	0x0
12	<b>GPIO12</b>	只读	0x0
11	<b>GPIO11</b>	只读	0x0
10	<b>GPIO10</b>	只读	0x0
9	<b>GPIO9</b>	只读	0x0
8	<b>GPIO8</b>	只读	0x0
7	<b>GPIO7</b>	只读	0x0
6	<b>GPIO6</b>	只读	0x0
5	<b>GPIO5</b>	只读	0x0
4	<b>GPIO4</b>	只读	0x0
3	<b>GPIO3</b>	只读	0x0
2	<b>GPIO2</b>	只读	0x0
1	<b>GPIO1</b>	只读	0x0

位	描述	类型	复位
0	<b>GPIO0</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROC1\_SECURE1寄存器

偏移: 0x214

表751.  
IRQSUMMARY\_PROC1  
\_SECURE1寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>GPIO47</b>	只读	0x0
14	<b>GPIO46</b>	只读	0x0
13	<b>GPIO45</b>	只读	0x0
12	<b>GPIO44</b>	只读	0x0
11	<b>GPIO43</b>	只读	0x0
10	<b>GPIO42</b>	只读	0x0
9	<b>GPIO41</b>	只读	0x0
8	<b>GPIO40</b>	只读	0x0
7	<b>GPIO39</b>	只读	0x0
6	<b>GPIO38</b>	只读	0x0
5	<b>GPIO37</b>	只读	0x0
4	<b>GPIO36</b>	只读	0x0
3	<b>GPIO35</b>	只读	0x0
2	<b>GPIO34</b>	只读	0x0
1	<b>GPIO33</b>	只读	0x0
0	<b>GPIO32</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROC1\_NONSECURE0寄存器

偏移: 0x218

表752.  
IRQSUMMARY\_PROC1  
\_NONSECURE0  
寄存器

位	描述	类型	复位
31	<b>GPIO31</b>	只读	0x0
30	<b>GPIO30</b>	只读	0x0
29	<b>GPIO29</b>	只读	0x0
28	<b>GPIO28</b>	只读	0x0
27	<b>GPIO27</b>	只读	0x0
26	<b>GPIO26</b>	只读	0x0
25	<b>GPIO25</b>	只读	0x0
24	<b>GPIO24</b>	只读	0x0
23	<b>GPIO23</b>	只读	0x0
22	<b>GPIO22</b>	只读	0x0

位	描述	类型	复位
21	<b>GPIO21</b>	只读	0x0
20	<b>GPIO20</b>	只读	0x0
19	<b>GPIO19</b>	只读	0x0
18	<b>GPIO18</b>	只读	0x0
17	<b>GPIO17</b>	只读	0x0
16	<b>GPIO16</b>	只读	0x0
15	<b>GPIO15</b>	只读	0x0
14	<b>GPIO14</b>	只读	0x0
13	<b>GPIO13</b>	只读	0x0
12	<b>GPIO12</b>	只读	0x0
11	<b>GPIO11</b>	只读	0x0
10	<b>GPIO10</b>	只读	0x0
9	<b>GPIO9</b>	只读	0x0
8	<b>GPIO8</b>	只读	0x0
7	<b>GPIO7</b>	只读	0x0
6	<b>GPIO6</b>	只读	0x0
5	<b>GPIO5</b>	只读	0x0
4	<b>GPIO4</b>	只读	0x0
3	<b>GPIO3</b>	只读	0x0
2	<b>GPIO2</b>	只读	0x0
1	<b>GPIO1</b>	只读	0x0
0	<b>GPIO0</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_PROC1\_NONSECURE1 寄存器

偏移: 0x21c

表 753  
IRQSUMMARY\_PROC1\_NONSECURE1 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>GPIO47</b>	只读	0x0
14	<b>GPIO46</b>	只读	0x0
13	<b>GPIO45</b>	只读	0x0
12	<b>GPIO44</b>	只读	0x0
11	<b>GPIO43</b>	只读	0x0
10	<b>GPIO42</b>	只读	0x0
9	<b>GPIO41</b>	只读	0x0
8	<b>GPIO40</b>	只读	0x0
7	<b>GPIO39</b>	只读	0x0

位	描述	类型	复位
6	<b>GPIO38</b>	只读	0x0
5	<b>GPIO37</b>	只读	0x0
4	<b>GPIO36</b>	只读	0x0
3	<b>GPIO35</b>	只读	0x0
2	<b>GPIO34</b>	只读	0x0
1	<b>GPIO33</b>	只读	0x0
0	<b>GPIO32</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_COMA\_WAKE\_SECURE0 寄存器

偏移: 0x220

表 754  
IRQSUMMARY\_COMA\_  
WAKE\_SECURE0  
寄存器

位	描述	类型	复位
31	<b>GPIO31</b>	只读	0x0
30	<b>GPIO30</b>	只读	0x0
29	<b>GPIO29</b>	只读	0x0
28	<b>GPIO28</b>	只读	0x0
27	<b>GPIO27</b>	只读	0x0
26	<b>GPIO26</b>	只读	0x0
25	<b>GPIO25</b>	只读	0x0
24	<b>GPIO24</b>	只读	0x0
23	<b>GPIO23</b>	只读	0x0
22	<b>GPIO22</b>	只读	0x0
21	<b>GPIO21</b>	只读	0x0
20	<b>GPIO20</b>	只读	0x0
19	<b>GPIO19</b>	只读	0x0
18	<b>GPIO18</b>	只读	0x0
17	<b>GPIO17</b>	只读	0x0
16	<b>GPIO16</b>	只读	0x0
15	<b>GPIO15</b>	只读	0x0
14	<b>GPIO14</b>	只读	0x0
13	<b>GPIO13</b>	只读	0x0
12	<b>GPIO12</b>	只读	0x0
11	<b>GPIO11</b>	只读	0x0
10	<b>GPIO10</b>	只读	0x0
9	<b>GPIO9</b>	只读	0x0
8	<b>GPIO8</b>	只读	0x0
7	<b>GPIO7</b>	只读	0x0

位	描述	类型	复位
6	<b>GPIO6</b>	只读	0x0
5	<b>GPIO5</b>	只读	0x0
4	<b>GPIO4</b>	只读	0x0
3	<b>GPIO3</b>	只读	0x0
2	<b>GPIO2</b>	只读	0x0
1	<b>GPIO1</b>	只读	0x0
0	<b>GPIO0</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_COMA\_WAKE\_SECURE1 寄存器

偏移: 0x224

表 755  
IRQSUMMARY\_COMA\_  
WAKE\_SECURE1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>GPIO47</b>	只读	0x0
14	<b>GPIO46</b>	只读	0x0
13	<b>GPIO45</b>	只读	0x0
12	<b>GPIO44</b>	只读	0x0
11	<b>GPIO43</b>	只读	0x0
10	<b>GPIO42</b>	只读	0x0
9	<b>GPIO41</b>	只读	0x0
8	<b>GPIO40</b>	只读	0x0
7	<b>GPIO39</b>	只读	0x0
6	<b>GPIO38</b>	只读	0x0
5	<b>GPIO37</b>	只读	0x0
4	<b>GPIO36</b>	只读	0x0
3	<b>GPIO35</b>	只读	0x0
2	<b>GPIO34</b>	只读	0x0
1	<b>GPIO33</b>	只读	0x0
0	<b>GPIO32</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_COMA\_WAKE\_NONSECURE0 寄存器

偏移: 0x228

表 756  
IRQSUMMARY\_COMA\_  
WAKE\_NONSECURE0  
寄存器

位	描述	类型	复位
31	<b>GPIO31</b>	只读	0x0
30	<b>GPIO30</b>	只读	0x0
29	<b>GPIO29</b>	只读	0x0
28	<b>GPIO28</b>	只读	0x0

位	描述	类型	复位
27	<b>GPIO27</b>	只读	0x0
26	<b>GPIO26</b>	只读	0x0
25	<b>GPIO25</b>	只读	0x0
24	<b>GPIO24</b>	只读	0x0
23	<b>GPIO23</b>	只读	0x0
22	<b>GPIO22</b>	只读	0x0
21	<b>GPIO21</b>	只读	0x0
20	<b>GPIO20</b>	只读	0x0
19	<b>GPIO19</b>	只读	0x0
18	<b>GPIO18</b>	只读	0x0
17	<b>GPIO17</b>	只读	0x0
16	<b>GPIO16</b>	只读	0x0
15	<b>GPIO15</b>	只读	0x0
14	<b>GPIO14</b>	只读	0x0
13	<b>GPIO13</b>	只读	0x0
12	<b>GPIO12</b>	只读	0x0
11	<b>GPIO11</b>	只读	0x0
10	<b>GPIO10</b>	只读	0x0
9	<b>GPIO9</b>	只读	0x0
8	<b>GPIO8</b>	只读	0x0
7	<b>GPIO7</b>	只读	0x0
6	<b>GPIO6</b>	只读	0x0
5	<b>GPIO5</b>	只读	0x0
4	<b>GPIO4</b>	只读	0x0
3	<b>GPIO3</b>	只读	0x0
2	<b>GPIO2</b>	只读	0x0
1	<b>GPIO1</b>	只读	0x0
0	<b>GPIO0</b>	只读	0x0

## IO\_BANK0: IRQSUMMARY\_COMA\_WAKE\_NONSECURE1 寄存器

偏移: 0x22c

表 757。  
IRQSUMMARY\_COMA\_WAKE\_NONSECURE1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>GPIO47</b>	只读	0x0
14	<b>GPIO46</b>	只读	0x0
13	<b>GPIO45</b>	只读	0x0

位	描述	类型	复位
12	<b>GPIO44</b>	只读	0x0
11	<b>GPIO43</b>	只读	0x0
10	<b>GPIO42</b>	只读	0x0
9	<b>GPIO41</b>	只读	0x0
8	<b>GPIO40</b>	只读	0x0
7	<b>GPIO39</b>	只读	0x0
6	<b>GPIO38</b>	只读	0x0
5	<b>GPIO37</b>	只读	0x0
4	<b>GPIO36</b>	只读	0x0
3	<b>GPIO35</b>	只读	0x0
2	<b>GPIO34</b>	只读	0x0
1	<b>GPIO33</b>	只读	0x0
0	<b>GPIO32</b>	只读	0x0

## IO\_BANK0: INTRO 寄存器

偏移: 0x230

### 描述

原始中断

表 758。INTRO  
寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	WC	0x0
30	<b>GPIO7_EDGE_LOW</b>	WC	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO7_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO6_EDGE_HIGH</b>	WC	0x0
26	<b>GPIO6_EDGE_LOW</b>	WC	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO6_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO5_EDGE_HIGH</b>	WC	0x0
22	<b>GPIO5_EDGE_LOW</b>	WC	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO5_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO4_EDGE_HIGH</b>	WC	0x0
18	<b>GPIO4_EDGE_LOW</b>	WC	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO4_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO3_EDGE_HIGH</b>	WC	0x0

位	描述	类型	复位
14	<b>GPIO3_EDGE_LOW</b>	WC	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO3_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO2_EDGE_HIGH</b>	WC	0x0
10	<b>GPIO2_EDGE_LOW</b>	WC	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO2_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO1_EDGE_HIGH</b>	WC	0x0
6	<b>GPIO1_EDGE_LOW</b>	WC	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO1_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO0_EDGE_HIGH</b>	WC	0x0
2	<b>GPIO0_EDGE_LOW</b>	WC	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO0_LEVEL_LOW</b>	只读	0x0

## IO\_BANK0：INTR1 寄存器

偏移: 0x234

### 描述

原始中断

表 759。INTR1  
寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	WC	0x0
30	<b>GPIO15_EDGE_LOW</b>	WC	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO15_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO14_EDGE_HIGH</b>	WC	0x0
26	<b>GPIO14_EDGE_LOW</b>	WC	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO14_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO13_EDGE_HIGH</b>	WC	0x0
22	<b>GPIO13_EDGE_LOW</b>	WC	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO13_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO12_EDGE_HIGH</b>	WC	0x0
18	<b>GPIO12_EDGE_LOW</b>	WC	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	只读	0x0

位	描述	类型	复位
16	<b>GPIO12_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO11_EDGE_HIGH</b>	WC	0x0
14	<b>GPIO11_EDGE_LOW</b>	WC	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO11_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO10_EDGE_HIGH</b>	WC	0x0
10	<b>GPIO10_EDGE_LOW</b>	WC	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO10_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO9_EDGE_HIGH</b>	WC	0x0
6	<b>GPIO9_EDGE_LOW</b>	WC	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO9_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO8_EDGE_HIGH</b>	WC	0x0
2	<b>GPIO8_EDGE_LOW</b>	WC	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO8_LEVEL_LOW</b>	只读	0x0

## IO\_BANK0：INTR2 寄存器

偏移: 0x238

### 描述

原始中断

表 760。INTR2  
寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	WC	0x0
30	<b>GPIO23_EDGE_LOW</b>	WC	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO23_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO22_EDGE_HIGH</b>	WC	0x0
26	<b>GPIO22_EDGE_LOW</b>	WC	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO22_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO21_EDGE_HIGH</b>	WC	0x0
22	<b>GPIO21_EDGE_LOW</b>	WC	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO21_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO20_EDGE_HIGH</b>	WC	0x0

位	描述	类型	复位
18	<b>GPIO20_EDGE_LOW</b>	WC	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO20_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO19_EDGE_HIGH</b>	WC	0x0
14	<b>GPIO19_EDGE_LOW</b>	WC	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO19_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO18_EDGE_HIGH</b>	WC	0x0
10	<b>GPIO18_EDGE_LOW</b>	WC	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO18_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO17_EDGE_HIGH</b>	WC	0x0
6	<b>GPIO17_EDGE_LOW</b>	WC	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO17_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO16_EDGE_HIGH</b>	WC	0x0
2	<b>GPIO16_EDGE_LOW</b>	WC	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO16_LEVEL_LOW</b>	只读	0x0

## IO\_BANK0：INTR3 寄存器

偏移: 0x23c

### 描述

原始中断

表 761。INTR3  
寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	WC	0x0
30	<b>GPIO31_EDGE_LOW</b>	WC	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO31_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO30_EDGE_HIGH</b>	WC	0x0
26	<b>GPIO30_EDGE_LOW</b>	WC	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO30_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO29_EDGE_HIGH</b>	WC	0x0
22	<b>GPIO29_EDGE_LOW</b>	WC	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	只读	0x0

位	描述	类型	复位
20	<b>GPIO29_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO28_EDGE_HIGH</b>	WC	0x0
18	<b>GPIO28_EDGE_LOW</b>	WC	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO28_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO27_EDGE_HIGH</b>	WC	0x0
14	<b>GPIO27_EDGE_LOW</b>	WC	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO27_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO26_EDGE_HIGH</b>	WC	0x0
10	<b>GPIO26_EDGE_LOW</b>	WC	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO26_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO25_EDGE_HIGH</b>	WC	0x0
6	<b>GPIO25_EDGE_LOW</b>	WC	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO25_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO24_EDGE_HIGH</b>	WC	0x0
2	<b>GPIO24_EDGE_LOW</b>	WC	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO24_LEVEL_LOW</b>	只读	0x0

## IO\_BANK0：INTR4 寄存器

偏移: 0x240

### 描述

原始中断

表 762。INTR4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	WC	0x0
30	<b>GPIO39_EDGE_LOW</b>	WC	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO39_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO38_EDGE_HIGH</b>	WC	0x0
26	<b>GPIO38_EDGE_LOW</b>	WC	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO38_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO37_EDGE_HIGH</b>	WC	0x0

位	描述	类型	复位
22	<b>GPIO37_EDGE_LOW</b>	WC	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO37_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO36_EDGE_HIGH</b>	WC	0x0
18	<b>GPIO36_EDGE_LOW</b>	WC	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO36_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO35_EDGE_HIGH</b>	WC	0x0
14	<b>GPIO35_EDGE_LOW</b>	WC	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO35_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO34_EDGE_HIGH</b>	WC	0x0
10	<b>GPIO34_EDGE_LOW</b>	WC	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO34_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO33_EDGE_HIGH</b>	WC	0x0
6	<b>GPIO33_EDGE_LOW</b>	WC	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO33_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO32_EDGE_HIGH</b>	WC	0x0
2	<b>GPIO32_EDGE_LOW</b>	WC	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO32_LEVEL_LOW</b>	只读	0x0

## IO\_BANK0: INTR5 寄存器

偏移: 0x244

### 描述

原始中断

表 763. INTR5  
寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	WC	0x0
30	<b>GPIO47_EDGE_LOW</b>	WC	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO47_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO46_EDGE_HIGH</b>	WC	0x0
26	<b>GPIO46_EDGE_LOW</b>	WC	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	只读	0x0

位	描述	类型	复位
24	<b>GPIO46_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO45_EDGE_HIGH</b>	WC	0x0
22	<b>GPIO45_EDGE_LOW</b>	WC	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO45_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO44_EDGE_HIGH</b>	WC	0x0
18	<b>GPIO44_EDGE_LOW</b>	WC	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO44_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO43_EDGE_HIGH</b>	WC	0x0
14	<b>GPIO43_EDGE_LOW</b>	WC	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO43_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO42_EDGE_HIGH</b>	WC	0x0
10	<b>GPIO42_EDGE_LOW</b>	WC	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO42_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO41_EDGE_HIGH</b>	WC	0x0
6	<b>GPIO41_EDGE_LOW</b>	WC	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO41_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO40_EDGE_HIGH</b>	WC	0x0
2	<b>GPIO40_EDGE_LOW</b>	WC	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO40_LEVEL_LOW</b>	只读	0x0

## IO\_BANK0: PROC0\_INTE0 寄存器

偏移: 0x248

### 描述

proc0的中断使能

表 764。  
PROC0\_INTE0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO7_EDGE_LOW</b>	读写	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO7_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO6_EDGE_HIGH</b>	读写	0x0

位	描述	类型	复位
26	<b>GPIO6_EDGE_LOW</b>	读写	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO6_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO5_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO5_EDGE_LOW</b>	读写	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO5_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO4_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO4_EDGE_LOW</b>	读写	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO4_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO3_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO3_EDGE_LOW</b>	读写	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO3_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO2_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO2_EDGE_LOW</b>	读写	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO2_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO1_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO1_EDGE_LOW</b>	读写	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO1_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO0_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO0_EDGE_LOW</b>	读写	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO0_LEVEL_LOW</b>	读写	0x0

## IO\_BANK0: PROC0\_INTE1 寄存器

偏移: 0x24c

### 描述

proc0的中断使能

表 765.  
PROC0\_INTE1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO15_EDGE_LOW</b>	读写	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	读写	0x0

位	描述	类型	复位
28	<b>GPIO15_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO14_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO14_EDGE_LOW</b>	读写	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO14_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO13_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO13_EDGE_LOW</b>	读写	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO13_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO12_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO12_EDGE_LOW</b>	读写	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO12_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO11_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO11_EDGE_LOW</b>	读写	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO11_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO10_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO10_EDGE_LOW</b>	读写	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO10_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO9_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO9_EDGE_LOW</b>	读写	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO9_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO8_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO8_EDGE_LOW</b>	读写	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO8_LEVEL_LOW</b>	读写	0x0

## IO\_BANK0: PROC0\_INTE2 寄存器

偏移: 0x250

### 描述

proc0的中断使能

表 766。  
PROC0\_INTE2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	读写	0x0

位	描述	类型	复位
30	<b>GPIO23_EDGE_LOW</b>	读写	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO23_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO22_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO22_EDGE_LOW</b>	读写	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO22_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO21_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO21_EDGE_LOW</b>	读写	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO21_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO20_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO20_EDGE_LOW</b>	读写	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO20_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO19_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO19_EDGE_LOW</b>	读写	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO19_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO18_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO18_EDGE_LOW</b>	读写	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO18_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO17_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO17_EDGE_LOW</b>	读写	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO17_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO16_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO16_EDGE_LOW</b>	读写	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO16_LEVEL_LOW</b>	读写	0x0

## IO\_BANK0: PROC0\_INTE3 寄存器

偏移: 0x254

### 描述

proc0的中断使能

表 767。  
PROC0\_INTE3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO31_EDGE_LOW</b>	读写	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO31_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO30_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO30_EDGE_LOW</b>	读写	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO30_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO29_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO29_EDGE_LOW</b>	读写	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO29_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO28_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO28_EDGE_LOW</b>	读写	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO28_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO27_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO27_EDGE_LOW</b>	读写	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO27_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO26_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO26_EDGE_LOW</b>	读写	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO26_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO25_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO25_EDGE_LOW</b>	读写	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO25_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO24_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO24_EDGE_LOW</b>	读写	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO24_LEVEL_LOW</b>	读写	0x0

## IO\_BANK0: PROC0\_INTE4 寄存器

偏移: 0x258

**描述**

proc0的中断使能

表 768  
PROC0\_INTE4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO39_EDGE_LOW</b>	读写	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO39_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO38_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO38_EDGE_LOW</b>	读写	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO38_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO37_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO37_EDGE_LOW</b>	读写	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO37_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO36_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO36_EDGE_LOW</b>	读写	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO36_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO35_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO35_EDGE_LOW</b>	读写	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO35_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO34_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO34_EDGE_LOW</b>	读写	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO34_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO33_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO33_EDGE_LOW</b>	读写	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO33_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO32_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO32_EDGE_LOW</b>	读写	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO32_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTE5 寄存器**

偏移: 0x25c

**描述**

proc0的中断使能

表 769  
PROC0\_INTE5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO47_EDGE_LOW</b>	读写	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO47_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO46_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO46_EDGE_LOW</b>	读写	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO46_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO45_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO45_EDGE_LOW</b>	读写	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO45_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO44_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO44_EDGE_LOW</b>	读写	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO44_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO43_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO43_EDGE_LOW</b>	读写	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO43_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO42_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO42_EDGE_LOW</b>	读写	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO42_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO41_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO41_EDGE_LOW</b>	读写	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO41_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO40_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO40_EDGE_LOW</b>	读写	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO40_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTFO 寄存器**

偏移: 0x260

**描述**

proc0 中断强制

表 770  
PROC0\_INTF0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO7_EDGE_LOW</b>	读写	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO7_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO6_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO6_EDGE_LOW</b>	读写	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO6_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO5_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO5_EDGE_LOW</b>	读写	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO5_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO4_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO4_EDGE_LOW</b>	读写	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO4_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO3_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO3_EDGE_LOW</b>	读写	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO3_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO2_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO2_EDGE_LOW</b>	读写	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO2_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO1_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO1_EDGE_LOW</b>	读写	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO1_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO0_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO0_EDGE_LOW</b>	读写	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO0_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTF1 寄存器**

偏移: 0x264

**描述**

proc0 中断强制

表 771  
PROC0\_INTF1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO15_EDGE_LOW</b>	读写	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO15_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO14_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO14_EDGE_LOW</b>	读写	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO14_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO13_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO13_EDGE_LOW</b>	读写	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO13_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO12_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO12_EDGE_LOW</b>	读写	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO12_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO11_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO11_EDGE_LOW</b>	读写	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO11_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO10_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO10_EDGE_LOW</b>	读写	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO10_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO9_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO9_EDGE_LOW</b>	读写	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO9_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO8_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO8_EDGE_LOW</b>	读写	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO8_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTF2 寄存器**

偏移: 0x268

**描述**

proc0 中断强制

表 772  
PROC0\_INTE2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO23_EDGE_LOW</b>	读写	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO23_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO22_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO22_EDGE_LOW</b>	读写	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO22_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO21_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO21_EDGE_LOW</b>	读写	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO21_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO20_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO20_EDGE_LOW</b>	读写	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO20_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO19_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO19_EDGE_LOW</b>	读写	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO19_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO18_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO18_EDGE_LOW</b>	读写	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO18_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO17_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO17_EDGE_LOW</b>	读写	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO17_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO16_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO16_EDGE_LOW</b>	读写	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO16_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTF3 寄存器**

偏移: 0x26c

**描述**

proc0 中断强制

表 773  
PROC0\_INTF3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO31_EDGE_LOW</b>	读写	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO31_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO30_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO30_EDGE_LOW</b>	读写	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO30_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO29_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO29_EDGE_LOW</b>	读写	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO29_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO28_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO28_EDGE_LOW</b>	读写	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO28_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO27_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO27_EDGE_LOW</b>	读写	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO27_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO26_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO26_EDGE_LOW</b>	读写	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO26_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO25_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO25_EDGE_LOW</b>	读写	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO25_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO24_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO24_EDGE_LOW</b>	读写	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO24_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTF4 寄存器**

偏移: 0x270

**描述**

proc0 中断强制

表 774  
PROC0\_INTF4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO39_EDGE_LOW</b>	读写	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO39_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO38_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO38_EDGE_LOW</b>	读写	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO38_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO37_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO37_EDGE_LOW</b>	读写	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO37_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO36_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO36_EDGE_LOW</b>	读写	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO36_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO35_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO35_EDGE_LOW</b>	读写	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO35_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO34_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO34_EDGE_LOW</b>	读写	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO34_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO33_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO33_EDGE_LOW</b>	读写	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO33_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO32_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO32_EDGE_LOW</b>	读写	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO32_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTF5 寄存器**

偏移: 0x274

**描述**

proc0 中断强制

表 775  
PROC0\_INTE5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO47_EDGE_LOW</b>	读写	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO47_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO46_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO46_EDGE_LOW</b>	读写	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO46_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO45_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO45_EDGE_LOW</b>	读写	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO45_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO44_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO44_EDGE_LOW</b>	读写	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO44_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO43_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO43_EDGE_LOW</b>	读写	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO43_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO42_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO42_EDGE_LOW</b>	读写	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO42_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO41_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO41_EDGE_LOW</b>	读写	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO41_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO40_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO40_EDGE_LOW</b>	读写	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO40_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC0\_INTS0 寄存器**

偏移: 0x278

**描述**

proc0 屏蔽及强制后的中断状态

表 776  
PROC0\_INTS0  
寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO7_EDGE_LOW</b>	只读	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO7_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO6_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO6_EDGE_LOW</b>	只读	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO6_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO5_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO5_EDGE_LOW</b>	只读	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO5_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO4_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO4_EDGE_LOW</b>	只读	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO4_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO3_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO3_EDGE_LOW</b>	只读	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO3_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO2_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO2_EDGE_LOW</b>	只读	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO2_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO1_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO1_EDGE_LOW</b>	只读	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO1_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO0_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO0_EDGE_LOW</b>	只读	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO0_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC0\_INTS1 寄存器**

偏移: 0x27c

**描述**

proc0 屏蔽及强制后的中断状态

表 777。  
PROC0\_INTS1  
寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO15_EDGE_LOW</b>	只读	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO15_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO14_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO14_EDGE_LOW</b>	只读	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO14_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO13_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO13_EDGE_LOW</b>	只读	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO13_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO12_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO12_EDGE_LOW</b>	只读	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO12_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO11_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO11_EDGE_LOW</b>	只读	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO11_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO10_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO10_EDGE_LOW</b>	只读	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO10_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO9_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO9_EDGE_LOW</b>	只读	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO9_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO8_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO8_EDGE_LOW</b>	只读	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO8_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC0\_INTS2 寄存器**

偏移: 0x280

**描述**

proc0 屏蔽及强制后的中断状态

表 778。  
PROC0\_INTS2  
寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO23_EDGE_LOW</b>	只读	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO23_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO22_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO22_EDGE_LOW</b>	只读	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO22_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO21_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO21_EDGE_LOW</b>	只读	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO21_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO20_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO20_EDGE_LOW</b>	只读	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO20_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO19_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO19_EDGE_LOW</b>	只读	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO19_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO18_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO18_EDGE_LOW</b>	只读	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO18_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO17_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO17_EDGE_LOW</b>	只读	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO17_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO16_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO16_EDGE_LOW</b>	只读	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO16_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC0\_INTS3 寄存器**

偏移: 0x284

**描述**

proc0 屏蔽及强制后的中断状态

表 779。  
PROC0\_INTS3  
寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO31_EDGE_LOW</b>	只读	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO31_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO30_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO30_EDGE_LOW</b>	只读	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO30_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO29_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO29_EDGE_LOW</b>	只读	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO29_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO28_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO28_EDGE_LOW</b>	只读	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO28_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO27_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO27_EDGE_LOW</b>	只读	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO27_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO26_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO26_EDGE_LOW</b>	只读	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO26_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO25_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO25_EDGE_LOW</b>	只读	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO25_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO24_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO24_EDGE_LOW</b>	只读	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO24_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC0\_INTS4 寄存器**

偏移: 0x288

**描述**

proc0 屏蔽及强制后的中断状态

表 780。  
PROC0\_INTS4  
寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO39_EDGE_LOW</b>	只读	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO39_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO38_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO38_EDGE_LOW</b>	只读	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO38_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO37_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO37_EDGE_LOW</b>	只读	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO37_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO36_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO36_EDGE_LOW</b>	只读	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO36_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO35_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO35_EDGE_LOW</b>	只读	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO35_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO34_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO34_EDGE_LOW</b>	只读	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO34_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO33_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO33_EDGE_LOW</b>	只读	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO33_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO32_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO32_EDGE_LOW</b>	只读	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO32_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC0\_INTS5 寄存器**

偏移: 0x28c

**描述**

proc0 屏蔽及强制后的中断状态

表 781.  
PROC0\_INTS5  
寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO47_EDGE_LOW</b>	只读	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO47_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO46_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO46_EDGE_LOW</b>	只读	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO46_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO45_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO45_EDGE_LOW</b>	只读	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO45_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO44_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO44_EDGE_LOW</b>	只读	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO44_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO43_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO43_EDGE_LOW</b>	只读	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO43_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO42_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO42_EDGE_LOW</b>	只读	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO42_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO41_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO41_EDGE_LOW</b>	只读	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO41_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO40_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO40_EDGE_LOW</b>	只读	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO40_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC1\_INTE0 寄存器**

偏移: 0x290

**描述**

proc1 中断使能

表 782.  
PROC1\_INTE0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO7_EDGE_LOW</b>	读写	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO7_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO6_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO6_EDGE_LOW</b>	读写	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO6_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO5_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO5_EDGE_LOW</b>	读写	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO5_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO4_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO4_EDGE_LOW</b>	读写	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO4_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO3_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO3_EDGE_LOW</b>	读写	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO3_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO2_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO2_EDGE_LOW</b>	读写	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO2_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO1_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO1_EDGE_LOW</b>	读写	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO1_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO0_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO0_EDGE_LOW</b>	读写	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO0_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTE1 寄存器**

偏移: 0x294

**描述**

proc1 中断使能

表 783.  
PROC1\_INTE1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO15_EDGE_LOW</b>	读写	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO15_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO14_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO14_EDGE_LOW</b>	读写	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO14_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO13_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO13_EDGE_LOW</b>	读写	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO13_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO12_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO12_EDGE_LOW</b>	读写	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO12_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO11_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO11_EDGE_LOW</b>	读写	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO11_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO10_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO10_EDGE_LOW</b>	读写	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO10_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO9_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO9_EDGE_LOW</b>	读写	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO9_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO8_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO8_EDGE_LOW</b>	读写	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO8_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTE2 寄存器**

偏移: 0x298

**描述**

proc1 中断使能

表 784.  
PROC1\_INTE2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO23_EDGE_LOW</b>	读写	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO23_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO22_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO22_EDGE_LOW</b>	读写	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO22_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO21_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO21_EDGE_LOW</b>	读写	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO21_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO20_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO20_EDGE_LOW</b>	读写	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO20_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO19_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO19_EDGE_LOW</b>	读写	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO19_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO18_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO18_EDGE_LOW</b>	读写	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO18_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO17_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO17_EDGE_LOW</b>	读写	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO17_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO16_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO16_EDGE_LOW</b>	读写	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO16_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTE3 寄存器**

偏移: 0x29c

**描述**

proc1 中断使能

表 785.  
PROC1\_INTE3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO31_EDGE_LOW</b>	读写	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO31_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO30_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO30_EDGE_LOW</b>	读写	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO30_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO29_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO29_EDGE_LOW</b>	读写	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO29_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO28_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO28_EDGE_LOW</b>	读写	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO28_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO27_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO27_EDGE_LOW</b>	读写	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO27_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO26_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO26_EDGE_LOW</b>	读写	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO26_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO25_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO25_EDGE_LOW</b>	读写	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO25_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO24_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO24_EDGE_LOW</b>	读写	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO24_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTE4 寄存器**

偏移: 0x2a0

**描述**

proc1 中断使能

表 786。  
PROC1\_INTE4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO39_EDGE_LOW</b>	读写	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO39_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO38_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO38_EDGE_LOW</b>	读写	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO38_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO37_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO37_EDGE_LOW</b>	读写	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO37_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO36_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO36_EDGE_LOW</b>	读写	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO36_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO35_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO35_EDGE_LOW</b>	读写	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO35_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO34_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO34_EDGE_LOW</b>	读写	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO34_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO33_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO33_EDGE_LOW</b>	读写	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO33_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO32_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO32_EDGE_LOW</b>	读写	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO32_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTE5 寄存器**

偏移: 0x2a4

**描述**

proc1 中断使能

表 787。  
PROC1\_INTE5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO47_EDGE_LOW</b>	读写	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO47_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO46_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO46_EDGE_LOW</b>	读写	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO46_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO45_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO45_EDGE_LOW</b>	读写	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO45_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO44_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO44_EDGE_LOW</b>	读写	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO44_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO43_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO43_EDGE_LOW</b>	读写	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO43_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO42_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO42_EDGE_LOW</b>	读写	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO42_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO41_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO41_EDGE_LOW</b>	读写	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO41_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO40_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO40_EDGE_LOW</b>	读写	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO40_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTFO 寄存器**

偏移: 0x2a8

**描述**

proc1 的中断强制

表 788。  
PROC1\_INTF0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO7_EDGE_LOW</b>	读写	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO7_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO6_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO6_EDGE_LOW</b>	读写	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO6_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO5_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO5_EDGE_LOW</b>	读写	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO5_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO4_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO4_EDGE_LOW</b>	读写	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO4_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO3_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO3_EDGE_LOW</b>	读写	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO3_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO2_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO2_EDGE_LOW</b>	读写	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO2_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO1_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO1_EDGE_LOW</b>	读写	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO1_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO0_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO0_EDGE_LOW</b>	读写	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO0_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTF1 寄存器**

偏移: 0x2ac

**描述**

proc1 的中断强制

表 789。  
PROC1\_INTF1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO15_EDGE_LOW</b>	读写	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO15_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO14_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO14_EDGE_LOW</b>	读写	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO14_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO13_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO13_EDGE_LOW</b>	读写	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO13_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO12_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO12_EDGE_LOW</b>	读写	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO12_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO11_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO11_EDGE_LOW</b>	读写	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO11_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO10_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO10_EDGE_LOW</b>	读写	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO10_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO9_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO9_EDGE_LOW</b>	读写	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO9_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO8_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO8_EDGE_LOW</b>	读写	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO8_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTF2 寄存器**

偏移量: 0x2b0

**描述**

proc1 的中断强制

表 790。  
PROC1\_INTF2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO23_EDGE_LOW</b>	读写	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO23_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO22_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO22_EDGE_LOW</b>	读写	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO22_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO21_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO21_EDGE_LOW</b>	读写	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO21_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO20_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO20_EDGE_LOW</b>	读写	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO20_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO19_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO19_EDGE_LOW</b>	读写	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO19_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO18_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO18_EDGE_LOW</b>	读写	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO18_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO17_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO17_EDGE_LOW</b>	读写	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO17_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO16_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO16_EDGE_LOW</b>	读写	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO16_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTF3 寄存器**

偏移量: 0x2b4

**描述**

proc1 的中断强制

表 791。  
PROC1\_INTF3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO31_EDGE_LOW</b>	读写	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO31_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO30_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO30_EDGE_LOW</b>	读写	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO30_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO29_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO29_EDGE_LOW</b>	读写	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO29_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO28_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO28_EDGE_LOW</b>	读写	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO28_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO27_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO27_EDGE_LOW</b>	读写	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO27_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO26_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO26_EDGE_LOW</b>	读写	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO26_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO25_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO25_EDGE_LOW</b>	读写	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO25_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO24_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO24_EDGE_LOW</b>	读写	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO24_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTF4 寄存器**

偏移量: 0x2b8

**描述**

proc1 的中断强制

表 792。  
PROC1\_INTF4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO39_EDGE_LOW</b>	读写	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO39_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO38_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO38_EDGE_LOW</b>	读写	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO38_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO37_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO37_EDGE_LOW</b>	读写	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO37_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO36_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO36_EDGE_LOW</b>	读写	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO36_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO35_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO35_EDGE_LOW</b>	读写	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO35_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO34_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO34_EDGE_LOW</b>	读写	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO34_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO33_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO33_EDGE_LOW</b>	读写	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO33_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO32_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO32_EDGE_LOW</b>	读写	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO32_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTF5 寄存器**

偏移量: 0x2bc

**描述**

proc1 的中断强制

表 793。  
PROC1\_INTE5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO47_EDGE_LOW</b>	读写	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO47_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO46_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO46_EDGE_LOW</b>	读写	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO46_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO45_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO45_EDGE_LOW</b>	读写	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO45_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO44_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO44_EDGE_LOW</b>	读写	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO44_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO43_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO43_EDGE_LOW</b>	读写	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO43_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO42_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO42_EDGE_LOW</b>	读写	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO42_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO41_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO41_EDGE_LOW</b>	读写	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO41_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO40_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO40_EDGE_LOW</b>	读写	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO40_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: PROC1\_INTS0 寄存器**

偏移量: 0x2c0

**描述**

proc1 掩码与强制后的中断状态

表 794.  
PROC1\_INTS0  
寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO7_EDGE_LOW</b>	只读	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO7_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO6_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO6_EDGE_LOW</b>	只读	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO6_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO5_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO5_EDGE_LOW</b>	只读	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO5_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO4_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO4_EDGE_LOW</b>	只读	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO4_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO3_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO3_EDGE_LOW</b>	只读	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO3_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO2_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO2_EDGE_LOW</b>	只读	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO2_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO1_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO1_EDGE_LOW</b>	只读	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO1_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO0_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO0_EDGE_LOW</b>	只读	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO0_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC1\_INTS1 寄存器**

偏移: 0x2c4

**描述**

proc1 掩码与强制后的中断状态

表 795.  
PROC1\_INTS1  
寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO15_EDGE_LOW</b>	只读	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO15_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO14_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO14_EDGE_LOW</b>	只读	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO14_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO13_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO13_EDGE_LOW</b>	只读	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO13_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO12_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO12_EDGE_LOW</b>	只读	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO12_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO11_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO11_EDGE_LOW</b>	只读	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO11_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO10_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO10_EDGE_LOW</b>	只读	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO10_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO9_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO9_EDGE_LOW</b>	只读	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO9_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO8_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO8_EDGE_LOW</b>	只读	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO8_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC1\_INTS2 寄存器**

偏移: 0x2c8

**描述**

proc1 掩码与强制后的中断状态

表 796.  
PROC1\_INTS2  
寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO23_EDGE_LOW</b>	只读	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO23_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO22_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO22_EDGE_LOW</b>	只读	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO22_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO21_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO21_EDGE_LOW</b>	只读	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO21_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO20_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO20_EDGE_LOW</b>	只读	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO20_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO19_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO19_EDGE_LOW</b>	只读	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO19_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO18_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO18_EDGE_LOW</b>	只读	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO18_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO17_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO17_EDGE_LOW</b>	只读	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO17_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO16_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO16_EDGE_LOW</b>	只读	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO16_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC1\_INTS3 寄存器**

偏移: 0x2cc

**描述**

proc1 掩码与强制后的中断状态

表 797.  
PROC1\_INTS3  
寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO31_EDGE_LOW</b>	只读	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO31_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO30_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO30_EDGE_LOW</b>	只读	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO30_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO29_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO29_EDGE_LOW</b>	只读	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO29_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO28_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO28_EDGE_LOW</b>	只读	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO28_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO27_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO27_EDGE_LOW</b>	只读	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO27_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO26_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO26_EDGE_LOW</b>	只读	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO26_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO25_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO25_EDGE_LOW</b>	只读	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO25_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO24_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO24_EDGE_LOW</b>	只读	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO24_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC1\_INTS4 寄存器**

偏移: 0x2d0

**描述**

proc1 掩码与强制后的中断状态

表 798.  
PROC1\_INTS4  
寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO39_EDGE_LOW</b>	只读	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO39_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO38_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO38_EDGE_LOW</b>	只读	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO38_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO37_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO37_EDGE_LOW</b>	只读	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO37_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO36_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO36_EDGE_LOW</b>	只读	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO36_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO35_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO35_EDGE_LOW</b>	只读	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO35_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO34_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO34_EDGE_LOW</b>	只读	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO34_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO33_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO33_EDGE_LOW</b>	只读	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO33_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO32_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO32_EDGE_LOW</b>	只读	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO32_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: PROC1\_INTS5 寄存器**

偏移量: 0x2d4

**描述**

proc1 掩码与强制后的中断状态

表 799。  
PROC1\_INTS5  
寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO47_EDGE_LOW</b>	只读	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO47_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO46_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO46_EDGE_LOW</b>	只读	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO46_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO45_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO45_EDGE_LOW</b>	只读	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO45_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO44_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO44_EDGE_LOW</b>	只读	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO44_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO43_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO43_EDGE_LOW</b>	只读	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO43_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO42_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO42_EDGE_LOW</b>	只读	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO42_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO41_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO41_EDGE_LOW</b>	只读	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO41_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO40_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO40_EDGE_LOW</b>	只读	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO40_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTE0 寄存器**

偏移量: 0x2d8

**描述**

dormant\_wake 的中断使能

表 800。  
DORMANT\_WAKE\_INT  
E0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO7_EDGE_LOW</b>	读写	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO7_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO6_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO6_EDGE_LOW</b>	读写	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO6_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO5_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO5_EDGE_LOW</b>	读写	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO5_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO4_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO4_EDGE_LOW</b>	读写	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO4_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO3_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO3_EDGE_LOW</b>	读写	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO3_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO2_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO2_EDGE_LOW</b>	读写	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO2_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO1_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO1_EDGE_LOW</b>	读写	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO1_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO0_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO0_EDGE_LOW</b>	读写	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO0_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTE1 寄存器**

偏移量: 0x2dc

**描述**

dormant\_wake 的中断使能

表 801。  
DORMANT\_WAKE\_INT  
E1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO15_EDGE_LOW</b>	读写	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO15_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO14_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO14_EDGE_LOW</b>	读写	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO14_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO13_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO13_EDGE_LOW</b>	读写	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO13_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO12_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO12_EDGE_LOW</b>	读写	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO12_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO11_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO11_EDGE_LOW</b>	读写	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO11_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO10_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO10_EDGE_LOW</b>	读写	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO10_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO9_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO9_EDGE_LOW</b>	读写	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO9_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO8_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO8_EDGE_LOW</b>	读写	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO8_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTE2 寄存器**

偏移量: 0x2e0

**描述**

dormant\_wake 的中断使能

表 802。  
DORMANT\_WAKE\_INT  
E2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO23_EDGE_LOW</b>	读写	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO23_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO22_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO22_EDGE_LOW</b>	读写	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO22_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO21_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO21_EDGE_LOW</b>	读写	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO21_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO20_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO20_EDGE_LOW</b>	读写	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO20_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO19_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO19_EDGE_LOW</b>	读写	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO19_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO18_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO18_EDGE_LOW</b>	读写	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO18_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO17_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO17_EDGE_LOW</b>	读写	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO17_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO16_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO16_EDGE_LOW</b>	读写	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO16_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTE3 寄存器**

偏移量: 0x2e4

**描述**

dormant\_wake 的中断使能

表 803.  
DORMANT\_WAKE\_INT  
E3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO31_EDGE_LOW</b>	读写	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO31_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO30_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO30_EDGE_LOW</b>	读写	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO30_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO29_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO29_EDGE_LOW</b>	读写	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO29_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO28_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO28_EDGE_LOW</b>	读写	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO28_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO27_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO27_EDGE_LOW</b>	读写	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO27_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO26_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO26_EDGE_LOW</b>	读写	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO26_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO25_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO25_EDGE_LOW</b>	读写	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO25_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO24_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO24_EDGE_LOW</b>	读写	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO24_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTE4 寄存器**

偏移：0x2e8

**描述**

dormant\_wake 的中断使能

表 804.  
DORMANT\_WAKE\_INT  
E4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO39_EDGE_LOW</b>	读写	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO39_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO38_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO38_EDGE_LOW</b>	读写	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO38_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO37_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO37_EDGE_LOW</b>	读写	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO37_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO36_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO36_EDGE_LOW</b>	读写	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO36_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO35_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO35_EDGE_LOW</b>	读写	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO35_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO34_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO34_EDGE_LOW</b>	读写	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO34_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO33_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO33_EDGE_LOW</b>	读写	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO33_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO32_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO32_EDGE_LOW</b>	读写	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO32_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTE5 寄存器**

偏移：0x2ec

**描述**

dormant\_wake 的中断使能

表 805.  
DORMANT\_WAKE\_INT  
E5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO47_EDGE_LOW</b>	读写	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO47_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO46_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO46_EDGE_LOW</b>	读写	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO46_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO45_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO45_EDGE_LOW</b>	读写	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO45_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO44_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO44_EDGE_LOW</b>	读写	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO44_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO43_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO43_EDGE_LOW</b>	读写	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO43_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO42_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO42_EDGE_LOW</b>	读写	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO42_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO41_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO41_EDGE_LOW</b>	读写	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO41_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO40_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO40_EDGE_LOW</b>	读写	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO40_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INFO 寄存器**

偏移：0x2f0

**描述**

dormant\_wake 的中断强制

表 806.  
DORMANT\_WAKE\_INT  
F0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO7_EDGE_LOW</b>	读写	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO7_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO6_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO6_EDGE_LOW</b>	读写	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO6_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO5_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO5_EDGE_LOW</b>	读写	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO5_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO4_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO4_EDGE_LOW</b>	读写	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO4_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO3_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO3_EDGE_LOW</b>	读写	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO3_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO2_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO2_EDGE_LOW</b>	读写	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO2_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO1_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO1_EDGE_LOW</b>	读写	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO1_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO0_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO0_EDGE_LOW</b>	读写	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO0_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTF1 寄存器**

偏移：0x2f4

**描述**

dormant\_wake 的中断强制

表 807.  
DORMANT\_WAKE\_INT  
F1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO15_EDGE_LOW</b>	读写	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO15_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO14_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO14_EDGE_LOW</b>	读写	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO14_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO13_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO13_EDGE_LOW</b>	读写	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO13_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO12_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO12_EDGE_LOW</b>	读写	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO12_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO11_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO11_EDGE_LOW</b>	读写	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO11_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO10_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO10_EDGE_LOW</b>	读写	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO10_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO9_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO9_EDGE_LOW</b>	读写	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO9_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO8_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO8_EDGE_LOW</b>	读写	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO8_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTF2 寄存器**

偏移: 0x2f8

**描述**

dormant\_wake 的中断强制

表 808  
DORMANT\_WAKE\_INT  
F2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO23_EDGE_LOW</b>	读写	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO23_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO22_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO22_EDGE_LOW</b>	读写	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO22_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO21_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO21_EDGE_LOW</b>	读写	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO21_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO20_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO20_EDGE_LOW</b>	读写	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO20_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO19_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO19_EDGE_LOW</b>	读写	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO19_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO18_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO18_EDGE_LOW</b>	读写	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO18_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO17_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO17_EDGE_LOW</b>	读写	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO17_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO16_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO16_EDGE_LOW</b>	读写	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO16_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTF3 寄存器**

偏移: 0x2fc

**描述**

dormant\_wake 的中断强制

表 809  
DORMANT\_WAKE\_INT  
F3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO31_EDGE_LOW</b>	读写	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO31_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO30_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO30_EDGE_LOW</b>	读写	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO30_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO29_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO29_EDGE_LOW</b>	读写	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO29_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO28_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO28_EDGE_LOW</b>	读写	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO28_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO27_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO27_EDGE_LOW</b>	读写	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO27_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO26_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO26_EDGE_LOW</b>	读写	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO26_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO25_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO25_EDGE_LOW</b>	读写	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO25_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO24_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO24_EDGE_LOW</b>	读写	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO24_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTF4 寄存器**

偏移: 0x300

**描述**

dormant\_wake 的中断强制

表 810  
DORMANT\_WAKE\_INT  
F4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO39_EDGE_LOW</b>	读写	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO39_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO38_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO38_EDGE_LOW</b>	读写	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO38_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO37_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO37_EDGE_LOW</b>	读写	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO37_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO36_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO36_EDGE_LOW</b>	读写	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO36_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO35_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO35_EDGE_LOW</b>	读写	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO35_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO34_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO34_EDGE_LOW</b>	读写	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO34_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO33_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO33_EDGE_LOW</b>	读写	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO33_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO32_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO32_EDGE_LOW</b>	读写	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO32_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTF5 寄存器**

偏移: 0x304

**描述**

dormant\_wake 的中断强制

表 811  
DORMANT\_WAKE\_INT  
F5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO47_EDGE_LOW</b>	读写	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO47_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO46_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO46_EDGE_LOW</b>	读写	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO46_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO45_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO45_EDGE_LOW</b>	读写	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO45_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO44_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO44_EDGE_LOW</b>	读写	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO44_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO43_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO43_EDGE_LOW</b>	读写	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO43_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO42_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO42_EDGE_LOW</b>	读写	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO42_LEVEL_LOW</b>	读写	0x0
7	<b>GPIO41_EDGE_HIGH</b>	读写	0x0
6	<b>GPIO41_EDGE_LOW</b>	读写	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	读写	0x0
4	<b>GPIO41_LEVEL_LOW</b>	读写	0x0
3	<b>GPIO40_EDGE_HIGH</b>	读写	0x0
2	<b>GPIO40_EDGE_LOW</b>	读写	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	读写	0x0
0	<b>GPIO40_LEVEL_LOW</b>	读写	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTS0 寄存器**

偏移: 0x308

**描述**

dormant\_wake 掩码与强制后的中断状态

表 812。  
DORMANT\_WAKE\_INT  
S0 寄存器

位	描述	类型	复位
31	<b>GPIO7_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO7_EDGE_LOW</b>	只读	0x0
29	<b>GPIO7_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO7_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO6_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO6_EDGE_LOW</b>	只读	0x0
25	<b>GPIO6_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO6_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO5_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO5_EDGE_LOW</b>	只读	0x0
21	<b>GPIO5_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO5_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO4_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO4_EDGE_LOW</b>	只读	0x0
17	<b>GPIO4_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO4_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO3_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO3_EDGE_LOW</b>	只读	0x0
13	<b>GPIO3_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO3_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO2_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO2_EDGE_LOW</b>	只读	0x0
9	<b>GPIO2_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO2_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO1_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO1_EDGE_LOW</b>	只读	0x0
5	<b>GPIO1_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO1_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO0_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO0_EDGE_LOW</b>	只读	0x0
1	<b>GPIO0_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO0_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTS1 寄存器**

偏移：0x30c

**描述**

dormant\_wake 掩码与强制后的中断状态

表 813。  
DORMANT\_WAKE\_INT  
S1 寄存器

位	描述	类型	复位
31	<b>GPIO15_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO15_EDGE_LOW</b>	只读	0x0
29	<b>GPIO15_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO15_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO14_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO14_EDGE_LOW</b>	只读	0x0
25	<b>GPIO14_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO14_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO13_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO13_EDGE_LOW</b>	只读	0x0
21	<b>GPIO13_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO13_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO12_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO12_EDGE_LOW</b>	只读	0x0
17	<b>GPIO12_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO12_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO11_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO11_EDGE_LOW</b>	只读	0x0
13	<b>GPIO11_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO11_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO10_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO10_EDGE_LOW</b>	只读	0x0
9	<b>GPIO10_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO10_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO9_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO9_EDGE_LOW</b>	只读	0x0
5	<b>GPIO9_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO9_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO8_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO8_EDGE_LOW</b>	只读	0x0
1	<b>GPIO8_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO8_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTS2 寄存器**

偏移: 0x310

**描述**

dormant\_wake 掩码与强制后的中断状态

表 814。  
DORMANT\_WAKE\_INT  
S2 寄存器

位	描述	类型	复位
31	<b>GPIO23_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO23_EDGE_LOW</b>	只读	0x0
29	<b>GPIO23_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO23_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO22_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO22_EDGE_LOW</b>	只读	0x0
25	<b>GPIO22_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO22_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO21_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO21_EDGE_LOW</b>	只读	0x0
21	<b>GPIO21_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO21_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO20_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO20_EDGE_LOW</b>	只读	0x0
17	<b>GPIO20_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO20_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO19_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO19_EDGE_LOW</b>	只读	0x0
13	<b>GPIO19_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO19_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO18_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO18_EDGE_LOW</b>	只读	0x0
9	<b>GPIO18_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO18_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO17_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO17_EDGE_LOW</b>	只读	0x0
5	<b>GPIO17_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO17_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO16_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO16_EDGE_LOW</b>	只读	0x0
1	<b>GPIO16_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO16_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTS3 寄存器**

偏移: 0x314

**描述**

dormant\_wake 掩码与强制后的中断状态

表 815。  
DORMANT\_WAKE\_INT  
S3 寄存器

位	描述	类型	复位
31	<b>GPIO31_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO31_EDGE_LOW</b>	只读	0x0
29	<b>GPIO31_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO31_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO30_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO30_EDGE_LOW</b>	只读	0x0
25	<b>GPIO30_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO30_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO29_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO29_EDGE_LOW</b>	只读	0x0
21	<b>GPIO29_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO29_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO28_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO28_EDGE_LOW</b>	只读	0x0
17	<b>GPIO28_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO28_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO27_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO27_EDGE_LOW</b>	只读	0x0
13	<b>GPIO27_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO27_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO26_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO26_EDGE_LOW</b>	只读	0x0
9	<b>GPIO26_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO26_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO25_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO25_EDGE_LOW</b>	只读	0x0
5	<b>GPIO25_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO25_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO24_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO24_EDGE_LOW</b>	只读	0x0
1	<b>GPIO24_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO24_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTS4 寄存器**

偏移: 0x318

**描述**

dormant\_wake 掩码与强制后的中断状态

表 816。  
DORMANT\_WAKE\_INT  
S4 寄存器

位	描述	类型	复位
31	<b>GPIO39_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO39_EDGE_LOW</b>	只读	0x0
29	<b>GPIO39_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO39_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO38_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO38_EDGE_LOW</b>	只读	0x0
25	<b>GPIO38_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO38_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO37_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO37_EDGE_LOW</b>	只读	0x0
21	<b>GPIO37_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO37_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO36_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO36_EDGE_LOW</b>	只读	0x0
17	<b>GPIO36_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO36_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO35_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO35_EDGE_LOW</b>	只读	0x0
13	<b>GPIO35_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO35_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO34_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO34_EDGE_LOW</b>	只读	0x0
9	<b>GPIO34_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO34_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO33_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO33_EDGE_LOW</b>	只读	0x0
5	<b>GPIO33_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO33_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO32_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO32_EDGE_LOW</b>	只读	0x0
1	<b>GPIO32_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO32_LEVEL_LOW</b>	只读	0x0

**IO\_BANK0: DORMANT\_WAKE\_INTS5 寄存器**

偏移: 0x31c

**描述**

dormant\_wake 掩码与强制后的中断状态

表 817。  
DORMANT\_WAKE\_INT  
S5 寄存器

位	描述	类型	复位
31	<b>GPIO47_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO47_EDGE_LOW</b>	只读	0x0
29	<b>GPIO47_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO47_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO46_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO46_EDGE_LOW</b>	只读	0x0
25	<b>GPIO46_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO46_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO45_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO45_EDGE_LOW</b>	只读	0x0
21	<b>GPIO45_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO45_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO44_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO44_EDGE_LOW</b>	只读	0x0
17	<b>GPIO44_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO44_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO43_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO43_EDGE_LOW</b>	只读	0x0
13	<b>GPIO43_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO43_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO42_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO42_EDGE_LOW</b>	只读	0x0
9	<b>GPIO42_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO42_LEVEL_LOW</b>	只读	0x0
7	<b>GPIO41_EDGE_HIGH</b>	只读	0x0
6	<b>GPIO41_EDGE_LOW</b>	只读	0x0
5	<b>GPIO41_LEVEL_HIGH</b>	只读	0x0
4	<b>GPIO41_LEVEL_LOW</b>	只读	0x0
3	<b>GPIO40_EDGE_HIGH</b>	只读	0x0
2	<b>GPIO40_EDGE_LOW</b>	只读	0x0
1	<b>GPIO40_LEVEL_HIGH</b>	只读	0x0
0	<b>GPIO40_LEVEL_LOW</b>	只读	0x0

### 9.11.2. IO - QSPI 银行

QSPI 银行 IO 寄存器起始地址为 `0x40030000` (在 SDK 中定义为 `IO_QSPI_BASE`)。

表 818。IO\_QSPI  
寄存器列表

偏移量	名称	说明
0x000	<code>USBPHY_DP_STATUS</code>	
0x004	<code>USBPHY_DP_CTRL</code>	
0x008	<code>USBPHY_DM_STATUS</code>	
0x00c	<code>USBPHY_DM_CTRL</code>	
0x010	<code>GPIO_QSPI_SCLK_STATUS</code>	
0x014	<code>GPIO_QSPI_SCLK_CTRL</code>	
0x018	<code>GPIO_QSPI_SS_STATUS</code>	
0x01c	<code>GPIO_QSPI_SS_CTRL</code>	
0x020	<code>GPIO_QSPI_SD0_STATUS</code>	
0x024	<code>GPIO_QSPI_SD0_CTRL</code>	
0x028	<code>GPIO_QSPI_SD1_STATUS</code>	
0x02c	<code>GPIO_QSPI_SD1_CTRL</code>	
0x030	<code>GPIO_QSPI_SD2_STATUS</code>	
0x034	<code>GPIO_QSPI_SD2_CTRL</code>	
0x038	<code>GPIO_QSPI_SD3_STATUS</code>	
0x03c	<code>GPIO_QSPI_SD3_CTRL</code>	
0x200	<code>IRQSUMMARY_PROC0_SECURE</code>	
0x204	<code>IRQSUMMARY_PROC0_NONSECURE</code>	
0x208	<code>IRQSUMMARY_PROC1_SECURE</code>	
0x20c	<code>IRQSUMMARY_PROC1_NONSECURE</code>	
0x210	<code>IRQSUMMARY_COMA_WAKE_SECURE</code>	
0x214	<code>IRQSUMMARY_COMA_WAKE_NONSECURE</code>	
0x218	中断	原始中断
0x21c	<code>PROC0_INTE</code>	proc0的中断使能
0x220	<code>PROC0_INTF</code>	proc0的中断强制
0x224	<code>PROC0_INTS</code>	proc0 屏蔽及强制后的中断状态
0x228	<code>PROC1_INTE</code>	proc1 中断使能
0x22c	<code>PROC1_INTF</code>	proc1 中断强制
0x230	<code>PROC1_INTS</code>	proc1 掩码与强制后的中断状态
0x234	<code>DORMANT_WAKE_INTE</code>	dormant_wake 的中断使能
0x238	<code>DORMANT_WAKE_INTF</code>	dormant_wake 的中断强制
0x23c	<code>DORMANT_WAKE_INTS</code>	dormant_wake 掩码与强制后的中断状态

## IO\_QSPI: USBPHY\_DP\_STATUS 寄存器

偏移: 0x000

表 819。  
USBPHY\_DP\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: USBPHY\_DP\_CTRL 寄存器

偏移量: 0x004

表 820。  
USBPHY\_DP\_CTRL  
寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		

位	描述	类型	复位
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 驱动输出由funcsel选择的外设信号		
	0x1 → INVERT: 驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW: 驱动输出低电平		
	0x3 → HIGH: 驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值:		
	0x02 → UART1_TX		
	0x03 → I2C0_SDA		
	0x05 → SIO_56		
	0x1f → NULL		

## IO\_QSPI: USBPHY\_DM\_STATUS 寄存器

偏移: 0x008

表 821。  
**USBPHY\_DM\_STATUS**  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: USBPHY\_DM\_CTRL 寄存器

偏移量: 0x00c

表 822。  
**USBPHY\_DM\_CTRL**  
寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OE OVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x02 → UART1_RX		
	0x03 → I2C0_SCL		
	0x05 → SIO_57		
	0x1f → NULL		

## IO\_QSPI: GPIO\_QSPI\_SCLK\_STATUS 寄存器

偏移量: 0x010

表823  
GPIO\_QSPI\_SCLK\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: GPIO\_QSPI\_SCLK\_CTRL寄存器

偏移: 0x014

表824  
GPIO\_QSPI\_SCLK\_CTRL  
寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → XIP_SCLK		
	0x02 → UART1_CTS		
	0x03 → I2C1_SDA		
	0x05 → SIO_58		
	0x0b → UART1_TX		
	0x1f → NULL		

## IO\_QSPI: GPIO\_QSPI\_SS\_STATUS 寄存器

偏移量: 0x018

表825  
GPIO\_QSPI\_SS\_STATUS  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: GPIO\_QSPI\_SS\_CTRL 寄存器

偏移量: 0x01c

表826  
GPIO\_QSPI\_SS\_CTRL  
寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		

位	描述	类型	复位
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → XIP_SS_N_0		
	0x02 → UART1_RTS		
	0x03 → I2C1_SCL		
	0x05 → SIO_59		
	0x0b → UART1_RX		
	0x1f → NULL		

## IO\_QSPI: GPIO\_QSPI\_SD0\_STATUS 寄存器

偏移: 0x020

表827  
GPIO\_QSPI\_SDO\_STA  
TUS寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: GPIO\_QSPI\_SDO\_CTRL寄存器

偏移: 0x024

表828  
GPIO\_QSPI\_SDO\_CTR  
L寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		

位	描述	类型	复位
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由funcsel选择的外设信号		
	0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → XIP_SDO		
	0x02 → UART0_TX		
	0x03 → I2C0_SDA		
	0x05 → SIO_60		
	0x1f → NULL		

## IO\_QSPI: GPIO\_QSPI\_SD1\_STATUS 寄存器

偏移：0x028

表829  
GPIO\_QSPI\_SD1\_STA  
TUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: GPIO\_QSPI\_SD1\_CTRL 寄存器

偏移量：0x02c

表830  
GPIO\_QSPI\_SD1\_CTR  
L 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		

位	描述	类型	复位
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OE OVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → XIP_SD1		
	0x02 → UART0_RX		
	0x03 → I2C0_SCL		
	0x05 → SIO_61		
	0x1f → NULL		

## IO\_QSPI: GPIO\_QSPI\_SD2\_STATUS 寄存器

偏移量: 0x030

表 831。  
GPIO\_QSPI\_SD2\_STA  
TUS 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: GPIO\_QSPI\_SD2\_CTRL 寄存器

偏移: 0x034

表 832。  
GPIO\_QSPI\_SD2\_CTR  
L 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转中断		
	0x1 → INVERT: 反转中断		
	0x2 → LOW: 使中断保持低电平		
	0x3 → HIGH: 使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 不反转外围输入		
	0x1 → INVERT: 反转外围输入		
	0x2 → LOW: 使外围输入保持低电平		
	0x3 → HIGH: 使外围输入保持高电平		
15:14	<b>OEOVER</b>	读写	0x0
	枚举值:		
	0x0 → NORMAL: 由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT: 由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE: 禁用输出		
	0x3 → ENABLE: 启用输出		
13:12	<b>OUTOVER</b>	读写	0x0

位	描述	类型	复位
	枚举值： 0x0 → NORMAL：驱动输出由funcsel选择的外设信号 0x1 → INVERT：驱动输出由funcsel选择的外设信号的反相信号 0x2 → LOW：驱动输出低电平 0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据gpio表选择引脚功能 31 == NULL	读写	0x1f
	枚举值： 0x00 → XIP_SD2 0x02 → UART0_CTS 0x03 → I2C1_SDA 0x05 → SIO_62 0x0b → UART0_TX 0x1f → NULL		

## IO\_QSPI: GPIO\_QSPI\_SD3\_STATUS 寄存器

偏移量: 0x038

表 833。  
*GPIO\_QSPI\_SD3\_STA*  
*TUS* 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>IRQTOPROC</b> : 覆盖后传递给处理器的中断	只读	0x0
25:18	保留。	-	-
17	<b>INFROMPAD</b> : 滤波与覆盖前的引脚输入信号	只读	0x0
16:14	保留。	-	-
13	<b>OETOPAD</b> : 寄存器覆盖后传递给引脚的输出使能	只读	0x0
12:10	保留。	-	-
9	<b>OUTTOPAD</b> : 寄存器覆盖后传递给引脚的输出信号	只读	0x0
8:0	保留。	-	-

## IO\_QSPI: GPIO\_QSPI\_SD3\_CTRL 寄存器

偏移量: 0x03c

表 834。  
*GPIO\_QSPI\_SD3\_CTR*  
*L* 寄存器

位	描述	类型	复位
31:30	保留。	-	-
29:28	<b>IRQOVER</b>	读写	0x0
	枚举值：		

位	描述	类型	复位
	0x0 → NORMAL：不反转中断		
	0x1 → INVERT：反转中断		
	0x2 → LOW：使中断保持低电平		
	0x3 → HIGH：使中断保持高电平		
27:18	保留。	-	-
17:16	<b>INOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：不反转外围输入		
	0x1 → INVERT：反转外围输入		
	0x2 → LOW：使外围输入保持低电平		
	0x3 → HIGH：使外围输入保持高电平		
15:14	<b>OE OVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：由 funcsel 选择的外设信号驱动输出使能		
	0x1 → INVERT：由 funcsel 选择的外设信号的反相信号驱动输出使能		
	0x2 → DISABLE：禁用输出		
	0x3 → ENABLE：启用输出		
13:12	<b>OUTOVER</b>	读写	0x0
	枚举值：		
	0x0 → NORMAL：驱动输出由 funcsel 选择的外设信号		
	0x1 → INVERT：驱动输出由 funcsel 选择的外设信号的反相信号		
	0x2 → LOW：驱动输出低电平		
	0x3 → HIGH：驱动输出高电平		
11:5	保留。	-	-
4:0	<b>FUNCSEL</b> : 0-31 → 根据 gpio 表选择引脚功能 31 == NULL	读写	0x1f
	枚举值：		
	0x00 → XIP_SD3		
	0x02 → UART0_RTS		
	0x03 → I2C1_SCL		
	0x05 → SIO_63		
	0x0b → UART0_RX		
	0x1f → NULL		

## IO\_QSPI: IRQSUMMARY\_PROC0\_SECURE 寄存器

偏移: 0x200

表 835。  
IRQSUMMARY\_PROC0\_SECURE 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>GPIO_QSPI_SD3</b>	只读	0x0
6	<b>GPIO_QSPI_SD2</b>	只读	0x0
5	<b>GPIO_QSPI_SD1</b>	只读	0x0
4	<b>GPIO_QSPI_SD0</b>	只读	0x0
3	<b>GPIO_QSPI_SS</b>	只读	0x0
2	<b>GPIO_QSPI_SCLK</b>	只读	0x0
1	<b>USBPHY_DM</b>	只读	0x0
0	<b>USBPHY_DP</b>	只读	0x0

## IO\_QSPI: IRQSUMMARY\_PROC0\_NONSECURE 寄存器

偏移: 0x204

表 836。  
IRQSUMMARY\_PROC0\_NONSECURE 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>GPIO_QSPI_SD3</b>	只读	0x0
6	<b>GPIO_QSPI_SD2</b>	只读	0x0
5	<b>GPIO_QSPI_SD1</b>	只读	0x0
4	<b>GPIO_QSPI_SD0</b>	只读	0x0
3	<b>GPIO_QSPI_SS</b>	只读	0x0
2	<b>GPIO_QSPI_SCLK</b>	只读	0x0
1	<b>USBPHY_DM</b>	只读	0x0
0	<b>USBPHY_DP</b>	只读	0x0

## IO\_QSPI: IRQSUMMARY\_PROC1\_SECURE 寄存器

偏移: 0x208

表 837。  
IRQSUMMARY\_PROC1\_SECURE 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>GPIO_QSPI_SD3</b>	只读	0x0
6	<b>GPIO_QSPI_SD2</b>	只读	0x0
5	<b>GPIO_QSPI_SD1</b>	只读	0x0
4	<b>GPIO_QSPI_SD0</b>	只读	0x0
3	<b>GPIO_QSPI_SS</b>	只读	0x0
2	<b>GPIO_QSPI_SCLK</b>	只读	0x0
1	<b>USBPHY_DM</b>	只读	0x0
0	<b>USBPHY_DP</b>	只读	0x0

## IO\_QSPI: IRQSUMMARY\_PROC1\_NONSECURE 寄存器

偏移: 0x20c

表838。  
IRQSUMMARY\_PROC1\_NONSECURE 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>GPIO_QSPI_SD3</b>	只读	0x0
6	<b>GPIO_QSPI_SD2</b>	只读	0x0
5	<b>GPIO_QSPI_SD1</b>	只读	0x0
4	<b>GPIO_QSPI_SD0</b>	只读	0x0
3	<b>GPIO_QSPI_SS</b>	只读	0x0
2	<b>GPIO_QSPI_SCLK</b>	只读	0x0
1	<b>USBPHY_DM</b>	只读	0x0
0	<b>USBPHY_DP</b>	只读	0x0

## IO\_QSPI: IRQSUMMARY\_COMA\_WAKE\_SECURE 寄存器

偏移: 0x210

表839。  
IRQSUMMARY\_COMA\_WAKE\_SECURE 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>GPIO_QSPI_SD3</b>	只读	0x0
6	<b>GPIO_QSPI_SD2</b>	只读	0x0
5	<b>GPIO_QSPI_SD1</b>	只读	0x0
4	<b>GPIO_QSPI_SD0</b>	只读	0x0
3	<b>GPIO_QSPI_SS</b>	只读	0x0
2	<b>GPIO_QSPI_SCLK</b>	只读	0x0
1	<b>USBPHY_DM</b>	只读	0x0
0	<b>USBPHY_DP</b>	只读	0x0

## IO\_QSPI: IRQSUMMARY\_COMA\_WAKE\_NONSECURE 寄存器

偏移: 0x214

表840。  
IRQSUMMARY\_COMA\_WAKE\_NONSECURE 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>GPIO_QSPI_SD3</b>	只读	0x0
6	<b>GPIO_QSPI_SD2</b>	只读	0x0
5	<b>GPIO_QSPI_SD1</b>	只读	0x0
4	<b>GPIO_QSPI_SD0</b>	只读	0x0
3	<b>GPIO_QSPI_SS</b>	只读	0x0
2	<b>GPIO_QSPI_SCLK</b>	只读	0x0
1	<b>USBPHY_DM</b>	只读	0x0

位	描述	类型	复位
0	USBPHY_DP	只读	0x0

## IO\_QSPI：INTR 寄存器

偏移：0x218

### 描述

原始中断

表841。INTR  
寄存器

位	描述	类型	复位
31	GPIO_QSPI_SD3_EDGE_HIGH	WC	0x0
30	GPIO_QSPI_SD3_EDGE_LOW	WC	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH	只读	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW	只读	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH	WC	0x0
26	GPIO_QSPI_SD2_EDGE_LOW	WC	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH	只读	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW	只读	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH	WC	0x0
22	GPIO_QSPI_SD1_EDGE_LOW	WC	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH	只读	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW	只读	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH	WC	0x0
18	GPIO_QSPI_SD0_EDGE_LOW	WC	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH	只读	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW	只读	0x0
15	GPIO_QSPI_SS_EDGE_HIGH	WC	0x0
14	GPIO_QSPI_SS_EDGE_LOW	WC	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH	只读	0x0
12	GPIO_QSPI_SS_LEVEL_LOW	只读	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH	WC	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW	WC	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH	只读	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW	只读	0x0
7	USBPHY_DM_EDGE_HIGH	WC	0x0
6	USBPHY_DM_EDGE_LOW	WC	0x0
5	USBPHY_DM_LEVEL_HIGH	只读	0x0
4	USBPHY_DM_LEVEL_LOW	只读	0x0
3	USBPHY_DP_EDGE_HIGH	WC	0x0

位	描述	类型	复位
2	<b>USBPHY_DP_EDGE_LOW</b>	WC	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	只读	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	只读	0x0

## IO\_QSPI: PROC0\_INTE 寄存器

偏移: 0x21c

### 描述

proc0的中断使能

表 842。  
PROC0\_INTE 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	读写	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	读写	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	读写	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	读写	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	读写	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	读写	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	读写	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	读写	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	读写	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	读写	0x0

位	描述	类型	复位
4	<b>USBPHY_DM_LEVEL_LOW</b>	读写	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	读写	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	读写	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	读写	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	读写	0x0

## IO\_QSPI: PROC0\_INTF 寄存器

偏移: 0x220

### 描述

proc0 中断强制

表 843。  
PROC0\_INTF 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	读写	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	读写	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	读写	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	读写	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	读写	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	读写	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	读写	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	读写	0x0

位	描述	类型	复位
6	<b>USBPHY_DM_EDGE_LOW</b>	读写	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	读写	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	读写	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	读写	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	读写	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	读写	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	读写	0x0

## IO\_QSPI: PROC0\_INTS 寄存器

偏移: 0x224

### 描述

proc0 屏蔽及强制后的中断状态

表 844。  
PROC0\_INTS 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	只读	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	只读	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	只读	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	只读	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	只读	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	只读	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	只读	0x0

位	描述	类型	复位
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	只读	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	只读	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	只读	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	只读	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	只读	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	只读	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	只读	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	只读	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	只读	0x0

## IO\_QSPI: PROC1\_INTE 寄存器

偏移: 0x228

### 描述

proc1 中断使能

表 845。  
PROC1\_INTE 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	读写	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	读写	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	读写	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	读写	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	读写	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	读写	0x0

位	描述	类型	复位
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	读写	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	读写	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	读写	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	读写	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	读写	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	读写	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	读写	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	读写	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	读写	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	读写	0x0

## IO\_QSPI: PROC1\_INTF 寄存器

偏移: 0x22c

### 描述

proc1 中断强制

表 846。  
PROC1\_INTF 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	读写	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	读写	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	读写	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	读写	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	读写	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	读写	0x0

位	描述	类型	复位
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	读写	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	读写	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	读写	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	读写	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	读写	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	读写	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	读写	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	读写	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	读写	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	读写	0x0

## IO\_QSPI: PROC1\_INTS 寄存器

偏移: 0x230

### 描述

proc1 掩码与强制后的中断状态

表 847。  
PROC1\_INTS 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	只读	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	只读	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	只读	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	只读	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	只读	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	只读	0x0

位	描述	类型	复位
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	只读	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	只读	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	只读	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	只读	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	只读	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	只读	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	只读	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	只读	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	只读	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	只读	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	只读	0x0

## IO\_QSPI: DORMANT\_WAKE\_INTE 寄存器

偏移: 0x234

### 描述

dormant\_wake 的中断使能

表 848。  
DORMANT\_WAKE\_INTE 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	读写	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	读写	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	读写	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	读写	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	读写	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	读写	0x0

位	描述	类型	复位
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	读写	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	读写	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	读写	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	读写	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	读写	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	读写	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	读写	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	读写	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	读写	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	读写	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	读写	0x0

## IO\_QSPI: DORMANT\_WAKE\_INTF 寄存器

偏移: 0x238

### 描述

dormant\_wake 的中断强制

表 849。  
DORMANT\_WAKE\_INTF 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	读写	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	读写	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	读写	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	读写	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	读写	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	读写	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	读写	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	读写	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	读写	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	读写	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	读写	0x0
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	读写	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	读写	0x0

位	描述	类型	复位
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	读写	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	读写	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	读写	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	读写	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	读写	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	读写	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	读写	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	读写	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	读写	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	读写	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	读写	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	读写	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	读写	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	读写	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	读写	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	读写	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	读写	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	读写	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	读写	0x0

## IO\_QSPI: DORMANT\_WAKE\_INTS 寄存器

偏移: 0x23c

### 描述

dormant\_wake 掩码与强制后的中断状态

表 850。  
DORMANT\_WAKE\_INT  
S 寄存器

位	描述	类型	复位
31	<b>GPIO_QSPI_SD3_EDGE_HIGH</b>	只读	0x0
30	<b>GPIO_QSPI_SD3_EDGE_LOW</b>	只读	0x0
29	<b>GPIO_QSPI_SD3_LEVEL_HIGH</b>	只读	0x0
28	<b>GPIO_QSPI_SD3_LEVEL_LOW</b>	只读	0x0
27	<b>GPIO_QSPI_SD2_EDGE_HIGH</b>	只读	0x0
26	<b>GPIO_QSPI_SD2_EDGE_LOW</b>	只读	0x0
25	<b>GPIO_QSPI_SD2_LEVEL_HIGH</b>	只读	0x0
24	<b>GPIO_QSPI_SD2_LEVEL_LOW</b>	只读	0x0
23	<b>GPIO_QSPI_SD1_EDGE_HIGH</b>	只读	0x0
22	<b>GPIO_QSPI_SD1_EDGE_LOW</b>	只读	0x0
21	<b>GPIO_QSPI_SD1_LEVEL_HIGH</b>	只读	0x0

位	描述	类型	复位
20	<b>GPIO_QSPI_SD1_LEVEL_LOW</b>	只读	0x0
19	<b>GPIO_QSPI_SD0_EDGE_HIGH</b>	只读	0x0
18	<b>GPIO_QSPI_SD0_EDGE_LOW</b>	只读	0x0
17	<b>GPIO_QSPI_SD0_LEVEL_HIGH</b>	只读	0x0
16	<b>GPIO_QSPI_SD0_LEVEL_LOW</b>	只读	0x0
15	<b>GPIO_QSPI_SS_EDGE_HIGH</b>	只读	0x0
14	<b>GPIO_QSPI_SS_EDGE_LOW</b>	只读	0x0
13	<b>GPIO_QSPI_SS_LEVEL_HIGH</b>	只读	0x0
12	<b>GPIO_QSPI_SS_LEVEL_LOW</b>	只读	0x0
11	<b>GPIO_QSPI_SCLK_EDGE_HIGH</b>	只读	0x0
10	<b>GPIO_QSPI_SCLK_EDGE_LOW</b>	只读	0x0
9	<b>GPIO_QSPI_SCLK_LEVEL_HIGH</b>	只读	0x0
8	<b>GPIO_QSPI_SCLK_LEVEL_LOW</b>	只读	0x0
7	<b>USBPHY_DM_EDGE_HIGH</b>	只读	0x0
6	<b>USBPHY_DM_EDGE_LOW</b>	只读	0x0
5	<b>USBPHY_DM_LEVEL_HIGH</b>	只读	0x0
4	<b>USBPHY_DM_LEVEL_LOW</b>	只读	0x0
3	<b>USBPHY_DP_EDGE_HIGH</b>	只读	0x0
2	<b>USBPHY_DP_EDGE_LOW</b>	只读	0x0
1	<b>USBPHY_DP_LEVEL_HIGH</b>	只读	0x0
0	<b>USBPHY_DP_LEVEL_LOW</b>	只读	0x0

### 9.11.3. 引脚控制 - 用户银行

用户银行垫控制寄存器起始基址为 `0x40038000` (在SDK中定义为PADS\_BANK0\_BASE)。

表851。 PADS\_B  
ANK0 寄存  
器列表

偏移量	名称	说明
0x00	<b>VOLTAGE_SELECT</b>	电压选择。单个银行控制
0x04	<b>GPIO0</b>	
0x08	<b>GPIO1</b>	
0x0c	<b>GPIO2</b>	
0x10	<b>GPIO3</b>	
0x14	<b>GPIO4</b>	
0x18	<b>GPIO5</b>	
0x1c	<b>GPIO6</b>	
0x20	<b>GPIO7</b>	
0x24	<b>GPIO8</b>	

偏移量	名称	说明
0x28	GPIO9	
0x2c	GPIO10	
0x30	GPIO11	
0x34	GPIO12	
0x38	GPIO13	
0x3c	GPIO14	
0x40	GPIO15	
0x44	GPIO16	
0x48	GPIO17	
0x4c	GPIO18	
0x50	GPIO19	
0x54	GPIO20	
0x58	GPIO21	
0x5c	GPIO22	
0x60	GPIO23	
0x64	GPIO24	
0x68	GPIO25	
0x6c	GPIO26	
0x70	GPIO27	
0x74	GPIO28	
0x78	GPIO29	
0x7c	GPIO30	
0x80	GPIO31	
0x84	GPIO32	
0x88	GPIO33	
0x8c	GPIO34	
0x90	GPIO35	
0x94	GPIO36	
0x98	GPIO37	
0x9c	GPIO38	
0xa0	GPIO39	
0xa4	GPIO40	
0xa8	GPIO41	
0xac	GPIO42	
0xb0	GPIO43	
0xb4	GPIO44	

偏移量	名称	说明
0xb8	GPIO45	
0xbc	GPIO46	
0xc0	GPIO47	
0xc4	SWCLK	
0xc8	SWD	

## PADS\_BANK0: VOLTAGE\_SELECT寄存器

偏移: 0x00

表852。  
VOLTAGE\_SELECT  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	电压选择。单个银行控制	读写	0x0
	枚举值：		
	0x0 → 3V3：设置电压为3.3V (DVDD >= 2V5)		
	0x1 → 1V8：设置电压为1.8V (DVDD ≤ 1V8)		

## PADS\_BANK0: GPIO0寄存器

偏移: 0x04

表853。GPIO0  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> ：垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> ：输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> ：输入使能	读写	0x0
5:4	<b>DRIVE</b> ：驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> ：使能上拉	读写	0x0
2	<b>PDE</b> ：使能下拉	读写	0x1
1	<b>SCHMITT</b> ：使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> ：转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0: GPIO1寄存器

偏移: 0x08

表854. GPIO1  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO2寄存器

偏移: 0x0c

表855. GPIO2  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO3寄存器

偏移: 0x10

表856. GPIO3  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO4寄存器

偏移：0x14

表857. GPIO4  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO5寄存器

偏移：0x18

表858. GPIO5  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO6寄存器

偏移：0x1c

表 859. GPIO6  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO7 寄存器

偏移：0x20

表 860. GPIO7  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO8 寄存器

偏移：0x24

表 861. GPIO8  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO9 寄存器

偏移量：0x28

表 862. GPIO9 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO10 寄存器

偏移量: 0x2c

表 863. GPIO10 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO11 寄存器

偏移量: 0x30

表 864. GPIO11  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO12 寄存器

偏移量: 0x34

表 865. GPIO12  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO13 寄存器

偏移量: 0x38

表 866. GPIO13  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO14 寄存器

偏移量: 0x3c

表 867. GPIO14  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO15 寄存器

偏移量: 0x40

表868. GPIO15  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO16寄存器

偏移：0x44

表869. GPIO16  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO17寄存器

偏移量：0x48

表870. GPIO17  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO18寄存器

偏移量: 0x4c

表871. GPIO18  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO19寄存器

偏移量: 0x50

表872. GPIO19  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO20寄存器

偏移量: 0x54

表873. GPIO20  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO21寄存器

偏移量: 0x58

表874. GPIO21  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO22寄存器

偏移量: 0x5c

表 875. GPIO22  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO23寄存器

偏移: 0x60

表 876. GPIO23 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO24寄存器

偏移: 0x64

表 877. GPIO24 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO25寄存器

偏移: 0x68

表 878. GPIO25  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO26寄存器

偏移: 0x6c

表 879. GPIO26  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO27寄存器

偏移量: 0x70

表 880. GPIO27  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO28寄存器

偏移量：0x74

表 881. GPIO28  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO29寄存器

偏移量：0x78

表 882. GPIO29  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO30寄存器

偏移: 0x7c

表 883. GPIO30  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO31寄存器

偏移: 0x80

表 884. GPIO31  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO32 寄存器

偏移: 0x84

表 885. GPIO32  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO33 寄存器

偏移: 0x88

表 886. GPIO33  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO34 寄存器

偏移量：0x8c

表 887. GPIO34  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO35 寄存器

偏移量：0x90

表 888. GPIO35  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO36 寄存器

偏移: 0x94

表 889. GPIO36  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO37 寄存器

偏移: 0x98

表 890。GPIO37  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO38 寄存器

偏移: 0x9c

表 891。GPIO38  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO39 寄存器

偏移: 0xa0

表 892. GPIO39  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO40 寄存器

偏移: 0xa4

表 893. GPIO40  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO41 寄存器

偏移: 0xa8

表 894. GPIO41  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO42 寄存器

偏移：0xac

表 895. GPIO42  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO43 寄存器

偏移：0xb0

表 896. GPIO43  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO44 寄存器

偏移：0xb4

表 897. GPIO44  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO45 寄存器

偏移：0xb8

表 898. GPIO45  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO46 寄存器

偏移：0xbc

表 899. GPIO46  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：GPIO47 寄存器

偏移：0xc0

表 900. GPI047  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x0
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：SWCLK 寄存器

偏移：0xc4

表 901. SWCLK  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x0
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x1
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x1
2	<b>PDE</b> : 使能下拉	读写	0x0
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_BANK0：SWD 寄存器

偏移量：0xc8

表 902. SWD 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x0
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x1
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值:		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x1
2	<b>PDE</b> : 使能下拉	读写	0x0
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快, 0 = 慢	读写	0x0

#### 9.11.4. 引脚控制 - QSPI 银行

QSPI Bank 组的 Pad 控制寄存器起始地址为 **0x40040000** (在 SDK 中定义为 PADS\_QSPI\_BASE)。

表 903. PADS\_Q SPI 寄存器列表

偏移量	名称	说明
0x00	<b>VOLTAGE_SELECT</b>	电压选择。单个银行控制
0x04	<b>GPIO_QSPI_SCLK</b>	
0x08	<b>GPIO_QSPI_SD0</b>	
0x0c	<b>GPIO_QSPI_SD1</b>	
0x10	<b>GPIO_QSPI_SD2</b>	
0x14	<b>GPIO_QSPI_SD3</b>	
0x18	<b>GPIO_QSPI_SS</b>	

#### PADS\_QSPI: VOLTAGE\_SELECT 寄存器

偏移: 0x00

表 904. VOLTAGE\_SELECT 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	电压选择。单个银行控制	读写	0x0
	枚举值:		
	0x0 → 3V3: 设置电压为3.3V (DVDD >= 2V5)		
	0x1 → 1V8: 设置电压为1.8V (DVDD <= 1V8)		

## PADS\_QSPI: GPIO\_QSPI\_SCLK 寄存器

偏移: 0x04

表 905.  
GPIO\_QSPI\_SCLK  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO:</b> 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD:</b> 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE:</b> 输入使能	读写	0x1
5:4	<b>DRIVE:</b> 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE:</b> 使能上拉	读写	0x0
2	<b>PDE:</b> 使能下拉	读写	0x1
1	<b>SCHMITT:</b> 使能施密特触发器	读写	0x1
0	<b>SLEWFAST:</b> 转换速率控制。1 = 快, 0 = 慢	读写	0x0

## PADS\_QSPI: GPIO\_QSPI\_SDO 寄存器

偏移: 0x08

表 906.  
GPIO\_QSPI\_SDO  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO:</b> 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD:</b> 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE:</b> 输入使能	读写	0x1
5:4	<b>DRIVE:</b> 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE:</b> 使能上拉	读写	0x0
2	<b>PDE:</b> 使能下拉	读写	0x1
1	<b>SCHMITT:</b> 使能施密特触发器	读写	0x1
0	<b>SLEWFAST:</b> 转换速率控制。1 = 快, 0 = 慢	读写	0x0

## PADS\_QSPI：GPIO\_QSPI\_SD1 寄存器

偏移: 0x0c

表 907.  
GPIO\_QSPI\_SD1  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x1
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x0
2	<b>PDE</b> : 使能下拉	读写	0x1
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快, 0 = 慢	读写	0x0

## PADS\_QSPI：GPIO\_QSPI\_SD2 寄存器

偏移: 0x10

表 908.  
GPIO\_QSPI\_SD2  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> : 垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> : 输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> : 输入使能	读写	0x1
5:4	<b>DRIVE</b> : 驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> : 使能上拉	读写	0x1
2	<b>PDE</b> : 使能下拉	读写	0x0
1	<b>SCHMITT</b> : 使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> : 转换速率控制。1 = 快, 0 = 慢	读写	0x0

## PADS\_QSPI：GPIO\_QSPI\_SD3寄存器

偏移：0x14

表909。  
GPIO\_QSPI\_SD3  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> ：垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> ：输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> ：输入使能	读写	0x1
5:4	<b>DRIVE</b> ：驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> ：使能上拉	读写	0x1
2	<b>PDE</b> ：使能下拉	读写	0x0
1	<b>SCHMITT</b> ：使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> ：转换速率控制。1 = 快，0 = 慢	读写	0x0

## PADS\_QSPI：GPIO\_QSPI\_SS寄存器

偏移：0x18

表910。  
GPIO\_QSPI\_SS  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ISO</b> ：垫隔离控制。垫一经软件配置后，应移除此项。	读写	0x1
7	<b>OD</b> ：输出禁用。优先于外设输出使能信号	读写	0x0
6	<b>IE</b> ：输入使能	读写	0x1
5:4	<b>DRIVE</b> ：驱动强度。	读写	0x1
	枚举值：		
	0x0 → 2MA		
	0x1 → 4MA		
	0x2 → 8MA		
	0x3 → 12MA		
3	<b>PUE</b> ：使能上拉	读写	0x1
2	<b>PDE</b> ：使能下拉	读写	0x0
1	<b>SCHMITT</b> ：使能施密特触发器	读写	0x1
0	<b>SLEWFAST</b> ：转换速率控制。1 = 快，0 = 慢	读写	0x0

# 第10章 安全性

本章介绍RP2350安全模型及其实现该模型的硬件。本章包含两个独立概述：一个针对Arm架构，另一个针对RISC-V架构。各架构具有不同的安全特性及bootrom支持级别。

## 10.1. 概述 (Arm)

RP2350为以下三项目的提供硬件及bootrom安全特性：

1. 防止未经授权的代码在设备上运行
2. 防止未经授权读取用户代码和数据
3. 隔离设备上同时运行的可信软件与不可信软件

本数据手册中提及的点 1称为**安全启动**。安全启动是第2点和第3点的先决条件，因为在设备上运行未经授权的代码将允许该代码访问设备内部结构。bootrom 的安全启动实现及相关硬件安全特性构成了安全RP2350应用的信任根。

bootrom 内容在设计阶段固化且不可更改。

本数据表中将第 2点称为加密启动。加密启动作为额外的保护层，使设备克隆、固件转储及逆向工程变得更加困难。加密启动通过在构建后阶段向二进制文件前附加签名的解密过程来实现。加密启动将解密密钥存储于设备上的一次性可编程 (OTP) 存储器中，且可在使用后被锁定。

第 3点允许应用程序实施内部安全边界，确保应用程序某部分遭到破坏时，不得访问诸如电压调节器或用于加密密钥的受保护OTP存储等关键硬件。

毛刺检测器和冗余协处理器等硬件特性可有效缓解常见类别的故障注入攻击，即使攻击者具备物理访问权限，也有助于维护启动完整性。

### 10.1.1. 安全启动

您可以永久更改空白的 RP2350 设备，以限制只执行您自己的代码。通过进一步修改，您可以撤销运行旧版本软件的权限。

RP2350 启动只读存储器使用加密签名来区分真伪二进制文件。签名是对二进制文件哈希后，使用用户私钥进行签名的结果。您可以将签名包含在为 RP2350 设备编译的二进制映像中。签名采用 SHA-256 哈希算法及 secp256k1 ECDSA 椭圆曲线密码算法对二进制文件进行身份验证。启动只读存储器通过以下步骤对二进制文件进行认证：

1. 加载二进制文件时，对映像代码和数据计算 SHA-256 哈希值。
2. 使用存储在映像中的用户公钥验证映像签名。
3. 将已包含且通过验证的签名（第 2 步）与第 1 步计算的二进制文件 SHA-256 哈希值进行比对。
4. 将映像的公钥与存储于 OTP 中的 SHA-256 密钥指纹进行核对。

如果两项检查均通过，bootrom 则假设持有由 OTP 公钥指纹注册的私钥者计算出了相同的 SHA-256 哈希值。基于哈希函数的性质，bootrom 假定该二进制内容自签名生成以来未被篡改。这证明该二进制文件为私钥所有者签署的真实文件，因此 bootrom 会考虑执行该二进制。

映像还可能包含防回滚版本号 (`rollback.major.minor`)，bootrom 会将其与

存储于 OTP 中的计数器进行比对。bootrom 拒绝启动回滚版本低于 OTP 计数器的映像，并在启动更高版本时自动递增 OTP 计数器。该机制在旧版本二进制存在已知漏洞时尤为重要，因为安装新版本会自动撤销回滚至旧版本的权限。

递增 `major` 及 `minor` 版本号能体现对较新、更高版本二进制的偏好，且不会阻止执行较旧、版本较低的二进制。有关只读存储器防回滚支持的更多讨论，请参见第 5.1.11 节。

RP2350 可以从以下任一源启动：

- 通过原地执行（XIP）直接从外部闪存启动
- 从外部闪存加载至 SRAM
- 从用户指定的一次性可编程（OTP）内容加载至 SRAM
- 通过 USB 或其他串行引导加载程序加载至 SRAM
- 通过调试器加载至 SRAM

当外部主机控制 RP2350 的处理器且可完全跳过只读存储器执行时，RP2350 对所有这些启动介质强制执行签名验证，但调试器除外。禁用调试是第 10.5 节所述安全启动启用程序的一部分。

虽然可以对闪存原地执行的二进制文件强制执行签名，但我们并不推荐这样做。因为使用该启动介质时，闪存内容可能在校验和执行间发生变化。例如，攻击者可能使用 FPGA 或其他微控制器模拟 QSPI 设备。建议将完整应用程序加载至 SRAM，并在执行前进行原地验证。RP2350 具备充足的 SRAM 容量，可满足大多数应用的需求。

纯软件的安全启动实现易受到故障注入攻击，尤其是在攻击者具备对设备的物理访问权限时，这种情况在嵌入式硬件中尤为常见。我们自主研发的 Pico 是一种广受欢迎的电压故障注入工具。RP2350 的故障检测器（第 10.9 节）与冗余协处理器（第 3.6.3 节）通过检测超出正常范围的操作并将系统安全地停止，减轻故障注入攻击，从而避免潜在的未经授权二进制程序启动。要启用故障检测器，请设置 CRIT1.GLITCH\_DETECTOR\_ENABLE 一次性可编程（OTP）标志位。

启动只读存储器（BootROM）始终使用冗余协处理器。

欲了解如何在空白 RP2350 设备上启用安全启动，请参阅第 10.5 节。

### 10.1.2. 加密启动

RP2350 含有 8 kB 的一次性可编程存储器（OTP），支持以 128 字节页面为粒度进行保护。该保护措施包括以下几种形式：

- 硬锁，永久撤销安全或非安全代码的读写权限。
- 软锁，仅在 OTP 块下一次复位之前撤销权限。

加密启动将解密密钥存储于 OTP，并通过软锁保护该密钥免受后续启动阶段的访问。

RP2350 支持将加密二进制文件从外部闪存加载至 SRAM，然后可在原位解密其内容。实现方式多样，但作为具体示例，本节介绍 SDK 与 `picotool` 提供的闪存驻留二进制加密支持。

1. 首先，开发者应将明文 SRAM 二进制文件转换为加密二进制文件。为加密您的二进制文件，SDK 在构建后执行以下步骤：

- a. 若未在构建过程中完成，请使用 **boot 私钥** 对有效载荷二进制文件进行签名。
- b. 使用 **加密密钥**（非私钥）对有效载荷二进制文件进行加密。
- c. 向二进制文件追加一个小型解密阶段，其中包含有效载荷的 `IMAGE_DEF` 的修改副本（原始版本因加密而不可读取）。
- d. 使用启动私钥对解密阶段及加密内容共同进行签名。

加密二进制文件作为打包的 RAM 二进制文件启动（见第 5.1.10 节），并在原位自行解密。启动

加密二进制文件时，启动只读存储器（bootrom）完成以下步骤：

1. 将整个加密二进制文件加载至SRAM。
2. 验证解密阶段的签名后，跳转至**解密阶段**，包含以下步骤：
  - a. 读取存储于OTP中的解密密钥（此阶段可能会软锁定该OTP页，直至下一次启动）。
  - b. 使用解密密钥对加密的二进制负载进行解密。
  - c. 在SRAM中解密的区域调用[chain\\_image\(\)](#)启动只读存储器API（见第5.4.8.2节）。
3. 以与验证解密阶段相同方式验证解密后的二进制负载，随后跳转执行该二进制文件。

解密阶段本身未加密，但已签名。以明文存储解密阶段不会带来额外风险，因其源代码为开源且经过严格审查。没有解密密钥，无法读取加密载荷。由于密钥仅存在于设备内部，静态分析加密二进制文件无法恢复该密钥。

重置OTP以重新开启软锁也将重置处理器。复位后，处理器会重新执行解密阶段，使用解密密钥重新锁定页面。BOOTDIS寄存器允许bootrom检测OTP复位并禁用看门狗及POWMAN启动向量。此举确保解密阶段不会被跳过，密钥始终受到保护。

#### 注意

解密阶段故意未包含在bootrom中，以便能够进行更新。bootrom仅处理公钥密码学，因此无需担心功率分析攻击，但该考虑不适用于解密阶段。针对功率分析的缓解措施需随着技术进步不断更新迭代。

该方案支持设计中，解密密钥仅对解密阶段可见。当解密密钥在运行时也需按需读取额外加密闪存内容时，处理器安全特性及OTP页面锁定可将密钥访问限制在极少数受信任代码内，如TF-M安全存储服务。

除了解密阶段提供的软件缓解措施外，RP2350 支持对其内部环形振荡器的频率控制进行随机化（第8.3节），以增加从电源踪迹恢复系统时钟的难度。

硬件不支持加密执行就地，但备用的32 MB 缓存XIP窗口（第4.4.1节）可以通过捕获缓存未命中并固定未命中地址，提供软件定义的执行就地功能。该功能可用于按需透明地解密外部闪存中的数据。

### 10.1.3. 隔离可信与不可信软件

在安全或安全关键应用中，必须将访问权限限制于必要的人员。例如，JPEG解码库不应能够访问核心电压调节器并将DV DD提升至3.3 V（除非您正在解压缩一个非常大的JPEG文件）。Cortex-M33处理器包含用于分隔两个执行上下文的硬件机制，分别称为安全和非安全，并在它们之间强制执行多项不变性，例如：

- 非安全代码无法访问安全内存
- 安全上下文无法执行非安全内存
- 非安全代码无法直接访问由安全代码管理的外设
- 非安全代码无法阻止安全中断被处理

通过减少可访问最关键硬件与数据的代码数量，可以降低意外将这些关键硬件与数据暴露给外部的风险。有关 Cortex-M3 3 实现此功能的高级说明，请参见第10.2节。详细信息请参考 Armv8-M 架构参考手册。

为了使安全软件与非安全软件的编程模型保持一致，并避免非安全代码的额外开销，RP2350 在整个系统中扩展了安全/非安全隔离。例如，DMA 通道可分配用于安全或非安全使用。借助此扩展隔离，非安全代码可通过 DMA 传输加速外设访问，同时不危及安全模型的约束（如非安全代码通过 DMA 读取安全内存）。

实现系统内安全与非安全分离的关键硬件特性包括：

- Cortex-M33对安全态与非安全态的实现（第10.2节）
- DMA对每通道安全状态匹配的实现（第10.7节）
- ACCESSCTRL实现的系统级总线访问过滤（第10.6节）
- 外围设备级过滤，例如SIO GPIO寄存器的每GPIO访问过滤（第3.1.1节）

## 10.2. 处理器安全特性（Arm）

RP2350上的Cortex-M33处理器配置了以下标准Arm安全特性：

- 支持Armv8-M安全扩展
- 8个安全归属单元（SAU）区域
- 8个安全和8个非安全内存保护单元（MPU）区域

这些特性在Armv8-M架构参考手册、Cortex-M33技术参考手册及本数据手册Cortex-M33章节（第3.7节）中均有详尽介绍。本节提供这些特性的高级概述，并描述RP2350中包含的实现定义归属单元。

### 10.2.1. 背景

RP2350上的Cortex-M33处理器支持Armv8-M安全扩展。处理器硬件维护两个独立的执行上下文，称为安全域和非安全域。对重要数据，如加密密钥，或硬件，如系统电压调节器的访问，可以限制在安全域内。将执行分离到这些域中，防止非安全执行干扰安全执行。当本数据手册使用大写术语 **Secure** 和 **Non-secure** 时，我们指代这两个Arm安全域及其相关的总线属性。

运行在非安全域中的代码不一定是恶意的。考虑到像USB这样复杂的协议和协议栈，其实现预期容易被利用且易发生致命崩溃。将此类软件限制在非安全域，有助于将关键软件与这些设计决策的后果隔离开来。例如，RP2350的bootrom在非安全域中运行所有USB代码，因此USB代码无需纳入bootrom关键部分的设计，如启动签名强制。

在任一时刻，实施安全扩展的 Armv8-M 处理器处于安全执行状态或非安全执行状态。根据当前状态，处理器限制可执行的内存区域及通过加载/存储指令可访问的内存区域。处理器所有的 AHB 访问均根据其来源状态进行标记，以便外设和系统总线结构本身可基于安全域过滤传输，例如，使用第10.6节所述的访问控制列表。

内部处理器外设——安全属性单元（SAU）从处理器视角定义哪些地址范围对安全域和非安全域可访问。SAU 可解码的不同地址范围数量有限，因此系统级总线过滤器用于将外设分配给安全域。

处理器通过状态间的特殊函数调用同步切换安全状态。当路由至安全域的中断发生时，若当前处于非安全状态，处理器也可异步切换安全状态，反之亦然（若已启用）。

RP2350上的两个Cortex-M33处理器均实现了安全扩展，因此每个处理器均保持其独立的安全

与非安全上下文。每个核心上的安全与非安全上下文可以相互通信，例如使用共享内存或安全/非安全SIO邮箱FIFO。如果核心对称使用（即共享的双核安全上下文和共享的双核非安全上下文），软件必须同步处理器SAU，以确保从一个核心非安全上下文可写的内存在另一个核心的安全上下文中不可执行。

支持与SAU相同区域形状和数量的DMA MPU，也必须与处理器SAU保持同步。

采用非对称的核心使用方式可能更为简便，将所有安全服务仅实现于一个核心。[FORCE\\_CORE\\_NS](#)寄存器可使所有核心1对系统总线的访问在安全结构和外设实施的安全筛选中，以及对于以安全/非安全方式划分的SIO寄存器组，均被视为非安全访问。

然而，这不影响PPB访问。这不影响核心1的内部状态，因此它仍可维持其自身的安全/非安全上下文。然而，系统硬件将所有核心1的访问视为非安全访问。

## 10.2.2. IDAU地址映射

Cortex-M33提供了一个实现定义的特征归属单元（IDAU）接口，允许系统实现者如Raspberry Pi Ltd扩展SAU所定义的安全归属映射。RP2350的IDAU为硬连线地址解码网络，不支持用户配置。其地址映射如下：

起始地址（十六进制）	结束地址（十六进制）	内容	IDAU属性
00000000	000042ff	Arm启动	豁免
00004300	00007dff	USB/RISC-V启动	非安全（指令抓取），豁免（加载/存储）
00007e00	00007ffff	Bootrom安全组	安全及非安全可调用
10000000	1fffffff	XIP	非安全
20000000	20081fff	SRAM	非安全
40000000	4fffffff	APB	豁免
50000000	5fffffff	AHB	豁免
d0000000	dfffffff	SIO	豁免

免检区域不会被处理器依据其当前安全状态进行检查。实际上，当处理器处于安全状态时，处理器将这些区域视为安全；而当处于非安全状态时，则视为非安全。

外设被标记为免检，因为您需要使用ACCESSCTRL（第10.6节）中的控制项将其分配到安全域。此方法避免依赖安全属性单元（SAU）区域，其范围过于有限，无法对外设进行有效分配，同时也消除了对独立的安全与非安全外设镜像的需求，防止了编程错误的发生。

SIO被标记为免检，因为其内部根据总线访问的安全属性在安全与非安全之间进行银行切换，该属性通常与处理器当前的安全状态对应。

鉴于外设为免检，RP2350通过物理断开总线，禁止处理器从外设处取指。即便覆盖默认MPU权限以允许执行，处理器仍无法从外设取指。免除区域允许安全和非安全访问，且TrustZone-M禁止非安全可写与安全可执行的组合，因此此限制为必要。对于bootrom，不适用相同考虑，因为只读存储器在物理上不可变更。

bootrom的第一部分为免除区域，因其包含预期由安全与非安全软件调用的例程，在某些情况下，非安全代码通过安全网关提升权限并不理想。例如bootrom `memcpy()`的实现。免除只读存储器区域内代码通过冗余协处理器的栈哨兵指令强化了对返回导向编程（ROP）攻击的防护能力。

在某一水位线之后，该水位线因只读存储器版本不同可能有所差异，仅用于指令取指目的，该只读存储器区域变为IDAU非安全区域。若非安全SAU区域覆盖bootrom（通常为在安全网关区域获得正确NSC属性），

该只读存储器部分对安全代码变为不可执行。因此，该bootrom部分未进行ROP强化。该ROM部分包括NSBOOT（含USB启动）实现及可用于在RISC-V处理器上模拟绝大多数bootrom功能的RISC-V Armv6-M仿真器。该区域仅通过指令侧IDAU查询实现：此为一项实现细节，旨在提升加载/存储IDAU查询的时序性能，无安全影响（鉴于掩码只读存储器本质不可写）——且 `tt` 指令对该区域不予识别。

引导只读存储器的最后512字节具有**安全、非安全可调用(NSC)**属性。这意味着该区域包含从非安全态调用安全态代码的入口点。请注意，若要使该IDAU定义的属性生效，相应范围内的SAU定义属性亦必须为NSC或更低级别。推荐配置为使用单个非安全SAU区域覆盖整个引导只读存储器。引导只读存储器在启用SAU的情况下退出至用户代码，SAU区域 7已激活并覆盖整个引导只读存储器。

XIP与SRAM在IDAU中被定义为非安全区，因其预期使用SAU划分。当SAU与IDAU属性不一致时，如果IDAU属性非免除，则处理器将按照“安全 > 非安全可调用 > 非安全”的顺序选取两者中级别更高者。

本表未列地址不会被系统AHB交叉总线解码，访问时将触发总线故障。在这些范围内，只读存储器的IDAU映射每32 kB镜像一次，直到 `0x0地址fffffff`。IDAU中剩余的地址均为非安全地址。

### 10.3. 概述 (RISC-V)

RP2350的bootrom未针对RISC-V处理器实现安全启动。尽管可以通过将安全启动代码存储在OTP中，并通过OTP中的BO OT\_FLAGS0行禁用其他启动介质来实现基于RISC-V的安全闪存启动，但RP2350的bootrom本身并不支持此功能。

RP2350上的RISC-V处理器实现了机器模式和用户模式执行，以及标准的物理内存保护单元（PMP），可用于强制执行内部安全或安全边界。详见第10.4节。

非处理器特定的硬件安全功能，如调试禁用OTP标志和故障检测器，在Arm和RISC-V之间功能一致。然而，冗余协处理器（RCP）因使用基于Cortex-M33的协处理器接口，RISC-V处理器无法访问。

### 10.4. 处理器安全特性 (RISC-V)

RP2350上的Hazard3处理器实现了以下标准RISC-V安全特性：

- 机器执行模式和用户执行模式（M模式和U模式）
- 物理内存保护单元（PMP）

M模式拥有对处理器内部状态寄存器的完全访问权限，而U模式则没有。处理器的总线访问会根据当前执行模式进行标记，并通过ACCESSCTRL总线过滤器进行过滤，如第10.6.2节所述。

处理器以M模式启动，并在发生任何陷阱（异常或中断）时进入M模式。仅当执行返回M模式指令 `mret` 且先前权限级为U模式时，处理器才切换至U模式。这意味着所有中断最初均定位于M模式，但可通过软件路由降级至U模式。鉴于RISC-V中栈由软件管理，完全分离这两种执行上下文需要软件配合，尽管存在足够的硬件钩子以实现此目的。有关RISC-V中断和异常的详细信息及其与核心权限级别的关系，请参阅第3.8.4节。

PMP是内置于每个RISC-V处理器中的内存保护单元，用于对每条指令执行地址及每个加载/存储地址进行权限区域列表的过滤。RP2350上的Hazard3实例各配置有8个PMP区域，支持32字节粒度以及自然对齐的2的幂次方区域。

此外，还存在3个PMP硬连线区域，为外设设置默认的用户模式读写权限，

并为只读存储器设置用户模式可读写执行（RWX）权限。这些区域编号为 8 至 10。由于编号较低的区域始终优先，任何动态配置的区域均可覆盖这些硬连线区域。

外设数量远远多于 PMP 区域数量。在典型使用场景中，程序员通常赋予这些外设统一的用户模式读写权限。由于硬连线区域成本远低于动态配置区域，使用硬连线区域更为高效。包含这些区域的原因是外设预期通过 ACCESSCTRL 进行权限分配，而非通过 PMP。硬连线区域的作用类似于 RP2350 Cortex-M IDAU 中的豁免区域。

结合ACCESSCTRL过滤器，这些PMP区域是区分U模式可访问地址与U模式不可访问地址的有效机制。Hazard3包含一项定制PMP功能，即PMPCFGM0寄存器，允许PMP设置 M模式权限及U模式权限，且无需锁定。

此功能对于防止对内存区域的意外（而非故意）访问十分有效。

## 10.5. 安全启动启用流程

启用安全启动：

1. 至少将一个公钥指纹写入OTP，起始地址为BOOTKEY0\_0。
2. 通过写入BOOT\_FLAGS1.KEY\_VALID将已写入的密钥标记为有效。
3. 可选地，建议通过写入BOOT\_FLAGS1.KEY\_INVALID将未使用的密钥标记为无效，以防止恶意行为者日后安装自有启动密钥。
  - KEY\_INVALID 优先于 KEY\_VALID，防止后续添加更多密钥。
  - 通过写入带有额外位的KEY\_INVALID，以便未来撤销密钥。
4. 通过写入CRIT1.DEBUG\_DISABLE、CRIT1.SECURE\_DEBUG\_DISABLE或安装调试密钥（参见第3.5.9.2节）来禁用调试功能。
5. 可选地，通过编程CRIT1.GLITCH\_DETECTOR\_ENABLE并在CRIT1.GLITCH\_DETECTOR\_SENS中设置所需灵敏度，以启用故障检测器（第10.9节）。
6. 在BOOT\_FLAGS0中禁用未使用的启动选项，如USB启动和UART启动。
7. 通过编程CRIT1.SECURE\_BOOT\_ENABLE启用安全启动功能。

### ● 警告

该操作不可逆。编程前，请确保使用正确且已正确哈希的公钥。[picotool](#)支持从标准PEM文件将密钥编程至OTP，并自动执行指纹哈希处理。编程错误的密钥将导致设备无法运行代码。

## 10.6. 访问控制

访问控制寄存器（ACCESSCTRL）定义访问GPIO及总线端点（如外设和存储设备）所需的权限。

对于每个总线端点（例如 [PIO0](#)），总线访问控制寄存器（如PIO0）控制哪些AHB5管理器可访问该端点及其访问的总线安全级别。该寄存器还涉及对该模块 [RESETS](#) 控制的访问权限。有关总线访问控制寄存器的完整说明，请参见第10.6.2节。

对于每个GPIO，包括QSPI和USB DP/DM引脚，GPIO\_NSmask0和GPIO\_NSmask1寄存器中的位可被设置，使该GPIO对安全域和非安全域均可访问，或清零使其仅对安全域可访问。这具有系统范围的影响，控制：

- 非安全SIO对GPIO的可见性

- 非安全代码对该GPIO的IO复用和引脚控制寄存器的访问
- 通过仅限安全总线访问的外设对GPIO的选择性访问权限

ACCESSCTRL寄存器始终可由任何安全或特权状态下的处理器完全读取，以便非安全软件能够枚举其被授权访问的硬件。然而，对ACCESSCTRL的写入受到严格限制。

非特权写入及来自DMA的写入将导致总线错误。来自非安全、特权（NSP）上下文的写入通常被忽略，唯一例外是总线访问控制寄存器中非安全、非特权（NSU）位的写入。仅当NSP位被设置时，NSU位才为非安全可写。

写操作可通过LOCK寄存器进一步锁定。此操作会导致特定管理器的写入被忽略。

详见第10.6.1节中的完整影响列表。

为降低意外写入风险，除GPIO\_NSMASK0和GPIO\_NSMASK1外，所有ACCESSCTRL寄存器的写入数据最高16位必须包含16位值0xacce。为此，应将值0xacce0000与写入数据执行按位或（OR）操作。原子SET/CLR/XOR别名写入同样必须携带该值。DMA写入被禁止，以避免配置错误的DMA通道意外清除权限。

### ！重要

当写入数据最高16位不等于0xacce时，写入操作将失败且返回总线错误（而非静默保持权限不变）。

最后，FORCE\_CORE\_NS寄存器使核心1的总线访问在系统级别表现为非安全访问。此方案支持所有安全服务运行于核心0，因此核心1不应能够访问安全硬件。

## 10.6.1. GPIO访问控制

GPIO非安全访问掩码寄存器GPIO\_NSmask0和GPIO\_NSmask1包含每个GPIO对应的一位。这两个寄存器的布局与SIO GPIO寄存器（第3.1.3节）布局一致，包括QSPI及USB DM/DP位的位置。当且仅当相关的GPIO\_NSmask位被设置时，非安全软件才能访问对应的GPIO。此举防止非安全软件干扰或监视安全软件使用的GPIO。

所有系统级GPIO控制寄存器，如IO和垫控制寄存器，均为安全及非安全代码共享。

但对这些寄存器的访问根据GPIO\_NSmask寄存器逐GPIO进行过滤。这意味着相同代码可在安全或非安全上下文中无需修改地运行，且只要已配置相应的GPIO安全掩码，安全软件无需为非安全GPIO访问实现任何接口。

设置GPIO\_NSmask位会对对应的GPIO产生以下影响：

- 相关SIO GPIO寄存器位（第3.1.3节）可通过对非安全SIO的总线访问进行访问。
  - 否则该位为只读零。
- 相关SIO GPIO寄存器位通过GPIO协处理器指令（第3.6.1节）对非安全代码开放访问权限。（第3.6.1节）。
  - 否则该位为只读零。
  - 非安全代码仅在NSACR中授予非安全权限访问协处理器0且在CPACR寄存器的非安全PPB实例中启用时，才能执行GPIO协处理器指令。
- 相关IO控制寄存器（第9.11.1节或第9.11.2节）对非安全代码开放访问权限。
  - 否则其为只读零。
- 该GPIO无法选择仅限安全外设的GPIO功能。
  - 试图选择此类外设时，将选择空功能（0x1f）替代。

- 如果在将该GPIO设置为非安全访问时选择了仅安全外设，则该选择将被更改为空功能。
- 相关引脚控制寄存器（第9.11.3节或第9.11.4节）变为非安全代码可访问。
  - 否则其为只读零。
- 该GPIO的中断将路由至非安全GPIO中断IO\_IRQ\_BANK0\_NS和IO\_IRQ\_QSPI\_NS，而非默认的安全中断IO\_IRQ\_BANK0和IO\_IRQ\_QSPI。（系统IRQ列表详见第3.2节。）
- 相关GPIO中断控制和状态位，例如PROC0\_INTS0，变为非安全代码可访问。
  - 否则，其值为只读零。
- 该GPIO可被非安全访问的PIO实例读取。
  - 否则，读取值为零。
  - 类似于SIO，PIO即使未选择功能，仍可监视GPIO，因此附加逻辑会将仅安全GPIO掩码至非安全访问的PIO实例。

**i 注意**

由于RP2350-E3，在RP2350A（QFN-60）上，访问 PADS\_BANK0 寄存器由GPIO\_NSMASK 中的错误位控制。在QFN-60封装上，必须禁用对PADS寄存器的非安全访问，并为非安全代码实现一个软件接口，以操作其分配的PADS寄存器。

### 10.6.2. 总线访问控制

总线访问控制寄存器定义了允许何种安全/非安全与特权/非特权组合访问每个下游总线端口。该机制还将外设分配到安全域。

此外，总线访问控制寄存器定义了允许哪些上游来源（处理器0/1、DMA或调试器）访问。

系统总线上的硬件过滤器（第2.1节）会针对每次访问检查其目标对应的权限列表。过滤器会阻止不符合该目标ACCESSC\_TRL寄存器中规定条件的访问；访问无法达到其目标，总线结构将直接返回总线错误。对目标寄存器无影响，且不返回任何数据。总线错误会导致违规处理器产生异常，或在违规的DMA通道上置位错误标志。

每个寄存器包含8位（例如ADC寄存器）。SP、SU、NSP和NSU位对应于发起总线传输的处理器安全状态或发起DMA通道的安全级别：

- SP位允许以下对象访问：
  - 运行于Arm处理器安全域内的特权软件
  - 运行于RISC-V处理器上的机器模式软件
  - 安全级别为 SP (3) 的DMA通道
- 必须设置 SU位且 SP位已设置，方可允许以下对象访问：
  - 运行于Arm处理器安全域内的用户（非特权）软件
  - 安全级别为 SU (2) 的DMA通道
- NSP位允许以下对象访问：
  - 运行于Arm处理器非安全域内的特权软件
  - 当设置FORCE\_CORE\_NS.CORE1时，运行于核心1安全域内的特权Arm软件
  - 当设置FORCE\_CORE\_NS.CORE1时，运行于核心1的机器模式RISC-V软件
  - 安全级别为 NSP (1) 的DMA通道

- 必须设置 **NSP**位和 **NSU**位，方可允许以下访问：
  - 运行于Arm处理器非安全域的用户（非特权）软件
  - 当设置**FORCE\_CORE\_NS.CORE1**时，运行于核心1的用户（非特权）Arm软件
  - 运行于RISC-V处理器上的用户模式软件
  - 安全级别为 **NSU** (0) 的DMA通道

**注意**

AHB Mem-AP访问的安全性及特权级别可配置，其总线安全性及特权级别与对应处理器中相应安全/特权上下文的加载/存储操作相同。每个Arm处理器配置有一个AHB Mem-AP。

**注意**

RISC-V调试模式的内存访问，其总线安全性及特权级别与运行于该处理器的机器模式软件相同；通过调试模块进行的RISC-V系统总线访问，其总线安全性及特权级别与运行于核心1的机器模式软件相同。

必须在相关安全/特权位之外，额外设置**DBG**、**DMA**、**CORE1**和**CORE0**位，方可允许特定总线管理器的访问。**DBG**位对应以下任一情况：

- 来自任一Arm处理器的AHB Mem-AP的访问
- 处于调试模式的任一RISC-V核心的访问
- 来自RISC-V系统总线的访问

将调试访问控制与软件驱动的处理器访问分离意味着，即使软件被锁定无法访问寄存器块，开发者仍可通过调试器访问该寄存器块。

大多数总线访问权限位均为安全、仅特权可写。唯一例外为**NSU**位，仅当同一寄存器中的**NSP**位被设置时，非安全特权上下文亦可写入该位。其目的在于，一旦安全域授予非安全访问权后，由非安全软件决定是否在非安全域内授予非特权访问。

### 10.6.2.1. 默认权限

大多数总线端点默认只允许来自任何主控设备的安全访问，但存在例外。以下端点默认对任何主控设备完全开放访问（任意安全级别与权限组合）（例如，因为它们预期由处理器的内部存储保护硬件进行划分）：

- 启动只读存储器（第4.1节）
- XIP（第4.4节）
- SRAM（第4.2节）
- 系统信息（SYSINFO，第12.15.1节）

以下端点默认只允许来自任何管理者的安全且特权访问（**SP**）：

- XIP\_AUX（DMA FIFO）（第4.4.3节）
- SHA-256（第12.13节）

以下端点默认只允许安全且特权访问（**SP**），且默认禁止DMA访问：

- 电源管理器（POWMAN，第6章），包括电源管理及电压调节器控制寄存器
- 真正随机数生成器（第12.12节）

- 时钟控制寄存器（第8.1节）
- XOSC（第8.2节）
- ROSC（第8.3节）
- SYSCFG（第12.15.2节）
- 锁相环（第8.6节）
- 时钟发生器（第8.5节）
- 看门狗（第12.9节）
- PSM（第7.4节）
- XIP控制寄存器（第4.4.5节）
- QMI（第12.14节）
- CoreSight 追踪 DMA FIFO
- CoreSight自托管调试窗口

任何未包含于上述列表的总线端点，默认仅允许安全访问，且可被任何管理器访问。

### 10.6.2.2 总线权限的其他影响

为避免诸如在非安全可访问GPIO上选择安全专用外设等矛盾配置，并提升安全与非安全软件间的可移植性，总线访问权限列表会传播至某些其他系统级硬件：

- RESETS模块（第7.5节）中某外设的复位控制仅当该外设自身为非安全可访问时，才为非安全可访问。
  - 非安全访问RESETS块本身也必须通过RESETS总线访问寄存器进行授权。
- 非安全不可访问的外设不能在非安全可访问的GPIO上选择功能。尝试这样操作将选择无效GPIO功能（`0x1f`）。
- 对非安全可访问和非安全不可访问的PIO块，均无法执行跨PIO操作，例如监测彼此的中断标志。
- 非安全可访问的PIO块无法读取仅限安全的GPIO。
- DMA通道低于最小已设有效权限位（`SP`置0时忽略 `SU`，`NSP`置0时忽略 `NSU`）将与该外设的DREQ信号断开。

### 10.6.2.3. 无总线访问控制的模块

ACCESSCTRL中有四个内存映射模块没有总线访问控制寄存器：

- Cortex-M的PPB为处理器内部组件，在安全/非安全域间内部切换共享。
- SIO内部通过安全/非安全访问方式实现银行切换。
- ACCESSCTRL始终为全局可读，并具备自身的内部写入过滤机制。
- 启动只读存储器仅支持安全访问且为硬连线连接。

### 10.6.3. 寄存器列表

ACCESSCTRL寄存器起始于基地址 `0x40060000`（SDK中定义为ACCESSCTRL\_BASE）。

表911。ACCES SCTRL 寄存器列表

偏移量	名称	说明
0x00	LOCK	<p>一旦LOCK位被写入1，ACCESSCTRL将静默忽略该主控的写入操作。LOCK仅可由安全的特权处理器或调试器写入。</p> <p>LOCK位仅当其值为零时方可写入。一旦设置，除非对ACCES SCTRL进行完全复位，否则LOCK位不可清除。</p> <p>设置LOCK位不会影响访问是否引发总线错误。非特权写入或DMA写入仍将引发总线错误。所有其他访问则继续不引发总线错误。</p>
0x04	FORCE_CORE_NS	<p>无论核心内部状态如何，强制核心1的总线访问始终为非安全访问。</p> <p>适用于将一核心指定为非安全核心的方案，因某些外围设备可能基于安全状态而非主控ID内部筛选单个寄存器。</p>
0x08	CFGRESET	<p>写入1以重置所有ACCESSCTRL配置，但不包括LOCK和FORC E_CORE_NS寄存器。</p> <p>该位用于RP2350只读存储器启动程序中，在启动过程中快速将ACCESSCTRL恢复至已知状态。</p> <p>请注意，与ACCESSCTRL中所有寄存器一样，当写入者对应的LOCK位被设置时，本寄存器不可写，因此，已被LOCK位锁定、无法访问ACCESSCTRL的主控不能使用CFGRESET寄存器更改内容。</p>
0x0c	GPIO_NSMASK0	<p>控制GPIO0...31是否允许非安全代码访问。</p> <p>仅允许安全、特权处理器或调试器写入。</p> <p>0 → 仅限安全访问</p> <p>1 → 安全与非安全访问</p>
0x10	GPIO_NSMASK1	控制GPIO32..47是否允许非安全代码访问，以及是否允许通过非安全SIO访问QSPI和USB位操作。仅允许安全、特权处理器或调试器写入。
0x14	只读存储器	控制只读存储器的访问权限。默认设置为完全开放访问。
0x18	XIP_MAIN	控制XIP_MAIN的访问权限。默认设置为完全开放访问。
0x1c	SRAM0	控制SRAM0的访问权限。默认设置为完全开放访问。
0x20	SRAM1	控制SRAM1的访问权限。默认设置为完全开放访问。
0x24	SRAM2	控制SRAM2的访问权限。默认设置为完全开放访问。
0x28	SRAM3	控制SRAM3的访问权限。默认设置为完全开放访问。
0x2c	SRAM4	控制SRAM4的访问权限。默认设置为完全开放访问。
0x30	SRAM5	控制SRAM5的访问权限。默认设置为完全开放访问。
0x34	SRAM6	控制SRAM6的访问权限。默认设置为完全开放访问。
0x38	SRAM7	控制SRAM7的访问权限。默认设置为完全开放访问。

偏移量	名称	说明
0x3c	SRAM8	控制SRAM8的访问权限。默认设置为完全开放访问。
0x40	SRAM9	控制SRAM9的访问权限。默认设置为完全开放访问。
0x44	DMA	控制DMA的访问权限。默认允许任何主设备进行安全访问。
0x48	USBCTRL	控制USBCTRL的访问权限。默认允许任何主设备进行安全访问。
0x4c	PIO0	控制PIO0的访问权限。默认允许任何主设备进行安全访问。
0x50	PIO1	控制PIO1的访问权限。默认允许任何主设备进行安全访问。
0x54	PIO2	控制对PIO2的访问。默认为任何主设备的安全访问。
0x58	CORESIGHT_TRACE	控制对CORESIGHT_TRACE的访问。默认为仅限安全特权处理器或调试访问。
0x5c	CORESIGHT_PERIPH	控制对CORESIGHT_PERIPH的访问。默认为仅限安全特权处理器或调试访问。
0x60	SYSINFO	控制对SYSINFO的访问。默认完全开放访问。
0x64	复位	控制对RESETS的访问。默认为任何主设备的安全访问。
0x68	IO_BANK0	控制对IO_BANK0的访问。默认为任何主设备的安全访问。
0x6c	IO_BANK1	控制对IO_BANK1的访问。默认为任何主设备的安全访问。
0x70	PADS_BANK0	控制对PADS_BANK0的访问。默认为任何主设备的安全访问。
0x74	PADS_QSPI	控制对PADS_QSPI的访问。默认为任何主设备的安全访问。
0x78	BUSCTRL	控制对BUSCTRL的访问。默认为任何主设备的安全访问。
0x7c	ADC	控制对ADC的访问。默认为任何主设备的安全访问。
0x80	HSTX	控制对HSTX的访问。默认为任何主设备的安全访问。
0x84	I2C0	控制对I2C0的访问。默认为任何主设备的安全访问。
0x88	I2C1	控制对I2C1的访问， 默认为任何主控的安全访问。
0x8c	PWM	控制对PWM的访问， 默认为任何主控的安全访问。
0x90	SPI0	控制对SPI0的访问， 默认为任何主控的安全访问。
0x94	SPI1	控制对SPI1的访问， 默认为任何主控的安全访问。

偏移量	名称	说明
0x98	TIMER0	控制对TIMER0的访问， 默认为任何主控的安全访问。
0x9c	TIMER1	控制对TIMER1的访问， 默认为任何主控的安全访问。
0xa0	UART0	控制对UART0的访问， 默认为任何主控的安全访问。
0xa4	UART1	控制对UART1的访问， 默认为任何主控的安全访问。
0xa8	OTP	控制对OTP的访问， 默认为任何主控的安全访问。
0xac	TBMAN	控制对TBMAN的访问， 默认为任何主控的安全访问。
0xb0	POWMAN	控制对POWMAN的访问， 默认为仅限安全、 特权处理器或调试访问。
0xb4	TRNG	控制对TRNG的访问， 默认为仅限安全、 特权处理器或调试访问。
0xb8	SHA256	控制对SHA256的访问， 默认为仅限安全特权访问。
0xbc	SYSCFG	控制对 SYSCFG 的访问。默认仅限安全态、 特权处理器或调试访问。
0xc0	时钟	控制对 CLOCKS 的访问。默认仅限安全态、 特权处理器或调试访问。
0xc4	XOSC	控制对 XOSC 的访问。默认仅限安全态、 特权处理器或调试访问。
0xc8	ROSC	控制对 ROSC 的访问。默认仅限安全态、 特权处理器或调试访问。
0xcc	PLL_SYS	控制对 PLL_SYS 的访问。默认仅限安全态、 特权处理器或调试访问。
0xd0	PLL_USB	控制对 PLL_USB 的访问。默认仅限安全态、 特权处理器或调试访问。
0xd4	TICKS	控制对 TICKS 的访问。默认仅限安全态、 特权处理器或调试访问。
0xd8	WATCHDOG	控制对 WATCHDOG 的访问。默认仅限安全态、 特权处理器或调试访问。
0xdc	PSM	控制对 PSM 的访问。默认仅限安全态、 特权处理器或调试访问。
0xe0	XIP_CTRL	控制对 XIP_CTRL 的访问。默认仅限安全态、 特权处理器或调试访问。
0xe4	XIP_QMI	控制对 XIP_QMI 的访问。默认仅限安全态、 特权处理器或调试访问。
0xe8	XIP_AUX	控制对 XIP_AUX 的访问。默认仅限安全且特权访问。

## ACCESSCTRL：LOCK 寄存器

偏移: 0x00

### 说明

一旦LOCK位被写入1，ACCESSCTRL将静默忽略该主控的写入操作。LOCK仅可由安全的特权处理器或调试器写入。

LOCK位仅当其值为零时方可写入。一旦设置，除非对ACCESSCTRL进行完全复位，否则LOCK位不可清除。

设置LOCK位不会影响访问是否引发总线错误。非特权写入或DMA写入仍将引发总线错误。所有其他访问则继续不引发总线错误。

表 912. *LOCK*  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>调试</b>	读写	0x0
2	<b>DMA</b>	只读	0x1
1	<b>CORE1</b>	读写	0x0
0	<b>CORE0</b>	读写	0x0

## ACCESSCTRL：FORCE\_CORE\_NS 寄存器

偏移: 0x04

### 描述

无论核心内部状态如何，强制核心1的总线访问始终为非安全访问。

适用于将某一核心指定为非安全核心的方案，因为部分外设可能基于安全状态内部筛选单个寄存器，但不基于主控 ID。

表 913.  
*FORCE\_CORE\_N*  
S 寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>CORE1</b>	读写	0x0
0	保留。	-	-

## ACCESSCTRL：CFGRESET 寄存器

偏移: 0x08

表 914。CFGRESET 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<p>写入1以重置所有ACCESSCTRL配置，但不包括LOCK和FORCE_CORE_NS寄存器。</p> <p>该位用于RP2350只读存储器启动程序中，在启动过程中快速将ACCESSCTRL恢复至已知状态。</p> <p>注意，与ACCESSCTRL中所有寄存器类似，当写入者对应的LOCK位被设置时，该寄存器不可写，因此已被锁定访问权限的主控无法使用CFGRESET寄存器修改其内容。</p>	SC	0x0

## ACCESSCTRL：GPIO\_NSmask0 寄存器

偏移: 0x0c

表 915。GPIO\_NSmask0 寄存器

位	描述	类型	复位
31:0	<p>控制GPIO0至GPIO31是否允许非安全代码访问。仅允许安全、特权处理器或调试器写入。</p> <p>0 → 仅限安全访问</p> <p>1 → 安全与非安全访问</p>	读写	0x00000000

## ACCESSCTRL：GPIO\_NSmask1 寄存器

偏移: 0x10

### 描述

控制GPIO32..47是否允许非安全代码访问，以及是否允许通过非安全SIO访问QSPI和USB位操作。仅允许安全、特权处理器或调试器写入。

表 916。GPIO\_NSmask1 寄存器

位	描述	类型	复位
31:28	<b>QSPI_SD</b>	读写	0x0
27	<b>QSPI_CSN</b>	读写	0x0
26	<b>QSPI_SCK</b>	读写	0x0
25	<b>USB_DM</b>	读写	0x0
24	<b>USB_DP</b>	读写	0x0
23:16	保留。	-	-
15:0	<b>GPIO</b>	读写	0x0000

## ACCESSCTRL：只读存储器寄存器

偏移: 0x14

### 说明

控制调试器、DMA、核心0和核心1是否能够访问只读存储器，以及它们能以何种安全性/权限等级进行访问。

默认情况下，访问权限完全开放。

此寄存器仅可由安全且具有特权权限的处理器或调试器写入，NSU位除外，

当设置NSP位时，NSU位可由非安全特权级写入。

表 917。只读存储器寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若值为1，调试器可根据本寄存器中SP/NSP/SU/NSU所允许的安全性与权限等级访问只读存储器。	读写	0x1
6	<b>DMA</b> : 若值为1，DMA可根据本寄存器中SP/NSP/SU/NSU所允许的安全性与权限等级访问只读存储器。	读写	0x1
5	<b>CORE1</b> : 若值为1，核心1可根据本寄存器中SP/NSP/SU/NSU所允许的安全性与权限等级访问只读存储器。	读写	0x1
4	<b>CORE0</b> : 如果为1，则核心0可访问只读存储器，访问权限依本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别。	读写	0x1
3	<b>SP</b> : 如果为1，则只读存储器可从安全、特权环境访问。	读写	0x1
2	<b>SU</b> : 如果为1且SP被设置，则只读存储器可从安全、无特权环境访问。	读写	0x1
1	<b>NSP</b> : 如果为1，则只读存储器可从非安全、特权环境访问。	读写	0x1
0	<b>NSU</b> : 如果为1且NSP被设置，则只读存储器可从非安全、无特权环境访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL: XIP\_MAIN 寄存器

偏移量：0x18

### 说明

控制调试器、DMA、核心0和核心1是否可访问XIP\_MAIN以及其访问的安全/特权级别。

默认情况下，访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表 918. XIP\_MAIN 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1，XIP_MAIN可被调试器访问，但仅限本寄存器中SP/NSP/SU/NSU允许的安全/权限级别。	读写	0x1
6	<b>DMA</b> : 若为1，XIP_MAIN可被DMA访问，但仅限本寄存器中SP/NSP/SU/NSU允许的安全/权限级别。	读写	0x1
5	<b>CORE1</b> : 若为1，XIP_MAIN可被核心1访问，但仅限本寄存器中SP/NSP/SU/NSU允许的安全/权限级别。	读写	0x1
4	<b>CORE0</b> : 若为1，XIP_MAIN可被核心0访问，但仅限本寄存器中SP/NSP/SU/NSU允许的安全/权限级别。	读写	0x1
3	<b>SP</b> : 若为1，XIP_MAIN可从安全、特权上下文访问。	读写	0x1
2	<b>SU</b> : 若为1且SP亦被置位，XIP_MAIN可从安全、非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 若为1，则可从非安全的特权上下文访问XIP_MAIN。	读写	0x1

位	描述	类型	复位
0	<b>NSU</b> : 若为1, 且NSP也已设置, 则可从非安全的非特权上下文访问XIP_MAIN。 ○ 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL: SRAM0寄存器

偏移: 0x1c

### 说明

控制调试器、DMA、核0和核1是否可以访问SRAM0, 以及其访问的安全级别与特权级别。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表919。SRAM0  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, 调试器可按本寄存器中SP/NSP/SU/NSU所允许的安全及特权级别访问SRAM0。	读写	0x1
6	<b>DMA</b> : 若为1, DMA可按本寄存器中SP/NSP/SU/NSU所允许的安全及特权级别访问SRAM0。	读写	0x1
5	<b>CORE1</b> : 若为1, 核1可按本寄存器中SP/NSP/SU/NSU所允许的安全及特权级别访问SRAM0。	读写	0x1
4	<b>CORE0</b> : 如果为1, SRAM0可由核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全与特权级别限制。	读写	0x1
3	<b>SP</b> : 如果为1, SRAM0可从安全且具有特权的上下文访问。	读写	0x1
2	<b>SU</b> : 如果为1且SP也被设置, SRAM0可从安全且无特权的上下文访问。	读写	0x1
1	<b>NSP</b> : 如果为1, SRAM0可从非安全且具有特权的上下文访问。	读写	0x1
0	<b>NSU</b> : 如果为1且NSP也被设置, SRAM0可从非安全且无特权的上下文访问。 ○ 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL: SRAM1寄存器

偏移: 0x20

### 说明

控制调试器、DMA、核心0和核心1是否可访问SRAM1及其访问的安全与特权级别。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表920. SRAM1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1, 调试器可访问SRAM1, 访问权限受本寄存器中SP/NSP/SU/N SU所允许的安全与特权级别限制。	读写	0x1
6	<b>DMA</b> : 如果为1, SRAM1可由DMA访问, 访问权限须符合本寄存器中SP/N SP/SU/NSU所允许的安全性及特权级别。	读写	0x1
5	<b>CORE1</b> : 如果为1, SRAM1可由核心1访问, 访问权限须符合本寄存器中S P/NSP/SU/NSU所允许的安全性及特权级别。	读写	0x1
4	<b>CORE0</b> : 如果为1, SRAM1可由核心0访问, 访问权限须符合本寄存器中S P/NSP/SU/NSU所允许的安全性及特权级别。	读写	0x1
3	<b>SP</b> : 如果为1, SRAM1可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 如果为1且同时设置了SP, SRAM1可从安全无特权上下文访问 。	读写	0x1
1	<b>NSP</b> : 如果为1, SRAM1可从非安全特权上下文访问。	读写	0x1
0	<b>NSU</b> : 如果为1且同时设置了NSP, SRAM1可从非安全无特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：SRAM2寄存器

偏移量: 0x24

### 说明

控制调试器、DMA、核心0和核心1是否能够访问SRAM2, 以及其访问的安全/特权等级。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表921. SRAM2  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若值为1, SRAM2可由调试器访问, 访问权限依本寄存器中SP/NSP/SU/ NSU所允许的安全/特权等级。	读写	0x1
6	<b>DMA</b> : 若值为1, SRAM2可由DMA访问, 访问权限依本寄存器中SP/NSP/S U/NSU所允许的安全/特权等级。	读写	0x1
5	<b>CORE1</b> : 若值为1, SRAM2可由核心1访问, 访问权限依本寄存器中SP/NS P/SU/NSU所允许的安全/特权等级。	读写	0x1
4	<b>CORE0</b> : 若值为1, SRAM2可由核心0访问, 访问权限依本寄存器中SP/NS P/SU/NSU所允许的安全/特权等级。	读写	0x1
3	<b>SP</b> : 当值为1时, SRAM2可从安全的特权上下文访问。	读写	0x1
2	<b>SU</b> : 当值为1且SP同时设置时, SRAM2可从安全的非特权上下文访 问。	读写	0x1
1	<b>NSP</b> : 当值为1时, SRAM2可从非安全的特权上下文访问。	读写	0x1

位	描述	类型	复位
0	<b>NSU</b> : 当值为1且NSP同时设置时, SRAM2可从非安全的非特权上下文访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL: SRAM3寄存器

偏移: 0x28

### 说明

控制调试器、DMA、核心0与核心1是否可访问SRAM3, 以及它们的安全与特权级别。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表922. SRAM3  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 当值为1时, SRAM3可被调试器访问, 访问权限由本寄存器中的SP/NSP/SU/NSU字段定义的安全及特权级别决定。	读写	0x1
6	<b>DMA</b> : 当值为1时, SRAM3可被DMA访问, 访问权限由本寄存器中的SP/NSP/SU/NSU字段定义的安全及特权级别决定。	读写	0x1
5	<b>CORE1</b> : 若值为1, 则在本寄存器中SP/NSP/SU/NSU许可的安全性和特权级别下, 核心1可访问SRAM3。	读写	0x1
4	<b>CORE0</b> : 若值为1, 则在本寄存器中SP/NSP/SU/NSU许可的安全性和特权级别下, 核心0可访问SRAM3。	读写	0x1
3	<b>SP</b> : 若值为1, 则SRAM3可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 若值为1, 且SP亦已设置, 则SRAM3可从安全非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 若值为1, 则SRAM3可从非安全特权上下文访问。	读写	0x1
0	<b>NSU</b> : 若值为1, 且NSP亦已设置, 则SRAM3可从非安全非特权上下文访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL: SRAM4寄存器

偏移: 0x2c

### 说明

控制调试器、DMA、核心0及核心1是否可以访问SRAM4, 以及访问时的安全性与特权级别。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表923. SRAM4  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若值为1, SRAM4可被调试器访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全与特权级别限制。	读写	0x1
6	<b>DMA</b> : 若值为1, SRAM4可被DMA访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全与特权级别限制。	读写	0x1
5	<b>CORE1</b> : 若值为1, SRAM4可被核心1访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全与特权级别限制。	读写	0x1
4	<b>CORE0</b> : 若值为1, SRAM4可被核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全与特权级别限制。	读写	0x1
3	<b>SP</b> : 若值为1, SRAM4可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 若值为1, 且同时设置SP, 则SRAM4可从安全非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 如果为1, SRAM4可从非安全特权上下文访问。	读写	0x1
0	<b>NSU</b> : 如果为1且NSP也已设置, SRAM4可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：SRAM5寄存器

偏移: 0x30

### 描述

控制调试器、DMA、核心0和核心1是否可访问SRAM5, 以及其安全/特权级别。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表924. SRAM5  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1, SRAM5可被调试器访问, 访问权限由本寄存器中的SP/NSP/SU/NSU安全及特权级别控制。	读写	0x1
6	<b>DMA</b> : 如果为1, SRAM5可被DMA访问, 访问权限由本寄存器中的SP/NSP/SU/NSU安全及特权级别控制。	读写	0x1
5	<b>CORE1</b> : 如果为1, SRAM5可被核心1访问, 访问权限由本寄存器中的SP/NSP/SU/NSU安全及特权级别控制。	读写	0x1
4	<b>CORE0</b> : 若为1, 则SRAM5可由核心0访问, 访问级别受本寄存器中SP/NSP/SU/NSU所规定的安全性及特权等级限制。	读写	0x1
3	<b>SP</b> : 若为1, 则SRAM5可由安全特权环境访问。	读写	0x1
2	<b>SU</b> : 若为1且SP亦已设置, 则SRAM5可由安全非特权环境访问。	读写	0x1
1	<b>NSP</b> : 若为1, 则SRAM5可由非安全特权环境访问。	读写	0x1

位	描述	类型	复位
0	<b>NSU</b> : 若为1且NSP亦已设置，则SRAM5可由非安全非特权环境访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：SRAM6寄存器

偏移: 0x34

### 描述

控制调试器、DMA、核心0及核心1对SRAM6的访问权限及其安全性/特权级别。

默认情况下，访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表925. SRAM6  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若值为1，SRAM6可由调试器访问，访问权限须符合本寄存器中SP/NSP/SU/NSU定义的安全性及权限级别。	读写	0x1
6	<b>DMA</b> : 若值为1，SRAM6可由DMA访问，访问权限须符合本寄存器中SP/N/SP/SU/NSU定义的安全性及权限级别。	读写	0x1
5	<b>CORE1</b> : 若值为1，SRAM6可由核心1访问，访问权限须符合本寄存器中SP/NSP/SU/NSU定义的安全性及权限级别。	读写	0x1
4	<b>CORE0</b> : 若值为1，SRAM6可由核心0访问，访问权限须符合本寄存器中SP/NSP/SU/NSU定义的安全性及权限级别。	读写	0x1
3	<b>SP</b> : 若值为1，SRAM6可在安全、特权上下文中访问。	读写	0x1
2	<b>SU</b> : 若值为1且SP同时被设置，SRAM6可在安全、非特权上下文中访问。	读写	0x1
1	<b>NSP</b> : 若为1，SRAM6可从非安全特权上下文访问。	读写	0x1
0	<b>NSU</b> : 若为1，且NSP同时设置，SRAM6可从非安全非特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：SRAM7寄存器

偏移: 0x38

### 描述

控制调试器、DMA、核心0和核心1是否能够访问SRAM7，以及它们的访问所允许的安全/特权级别。

默认情况下，访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表926. SRAM7  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可在本寄存器中由SP/NSP/SU/NSU允许的安全/特权级别访问SRAM7。	读写	0x1
6	<b>DMA:</b> 若为1, DMA可在本寄存器中由SP/NSP/SU/NSU允许的安全/特权级别访问SRAM7。	读写	0x1
5	<b>CORE1:</b> 若为1, 核心1可在本寄存器中由SP/NSP/SU/NSU允许的安全/特权级别访问SRAM7。	读写	0x1
4	<b>CORE0:</b> 如果为1, SRAM7可由核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU所规定的安全/特权级别限制。	读写	0x1
3	<b>SP:</b> 如果为1, SRAM7可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1且SP同时设置, SRAM7可从安全非特权上下文访问。	读写	0x1
1	<b>NSP:</b> 如果为1, SRAM7可从非安全特权上下文访问。	读写	0x1
0	<b>NSU:</b> 如果为1且NSP同时设置, SRAM7可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：SRAM8寄存器

偏移: 0x3c

### 描述

控制调试器、DMA、核心0及核心1对SRAM8的访问权限及相应安全/特权级别。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 927. SRAM8  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 如果为1, SRAM8可由调试器访问, 访问权限受本寄存器中SP/NSP/SU/NSU所规定的安全/特权级别限制。	读写	0x1
6	<b>DMA:</b> 若为1, 则SRAM8可由DMA访问, 访问权限受本寄存器中SP/NSP/SU/NSU所定义的安全与特权级别限制。	读写	0x1
5	<b>CORE1:</b> 若为1, 则SRAM8可由核心1访问, 访问权限受本寄存器中SP/NSP/SU/NSU所定义的安全与特权级别限制。	读写	0x1
4	<b>CORE0:</b> 若为1, 则SRAM8可由核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU所定义的安全与特权级别限制。	读写	0x1
3	<b>SP:</b> 若为1, 则SRAM8可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若为1, 且SP亦被设置, 则SRAM8可从安全非特权上下文访问。	读写	0x1
1	<b>NSP:</b> 若为1, 则SRAM8可从非安全特权上下文访问。	读写	0x1

位	描述	类型	复位
0	<b>NSU:</b> 若为1, 且NSP亦被设置, 则SRAM8可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：SRAM9寄存器

偏移量: 0x40

### 描述

控制调试器、DMA、核心0和核心1是否可以访问SRAM9, 以及它们可在本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别进行访问。

默认情况下, 访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 928。SRAM9  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问SRAM9。	读写	0x1
6	<b>DMA:</b> 若为1, DMA可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问SRAM9。	读写	0x1
5	<b>CORE1:</b> 若为1, 核心1可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问SRAM9。	读写	0x1
4	<b>CORE0:</b> 若为1, 核心0可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问SRAM9。	读写	0x1
3	<b>SP:</b> 如果为1, SRAM9可从安全且特权的上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1, 且SP也设置, SRAM9可从安全且非特权的上下文访问。	读写	0x1
1	<b>NSP:</b> 如果为1, SRAM9可从非安全且特权的上下文访问。	读写	0x1
0	<b>NSU:</b> 如果为1, 且NSP也设置, SRAM9可从非安全且非特权的上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL：DMA寄存器

偏移: 0x44

### 说明

控制调试器、DMA、核0及核1是否可以访问DMA, 以及其访问的安全性和特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表929. DMA  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 如果为1，调试器可访问DMA，访问权限由本寄存器中SP/NSP/SU/N SU字段控制的安全性和特权级别决定。	读写	0x1
6	<b>DMA:</b> 如果为1，DMA可访问本设备，访问权限由本寄存器中SP/NSP/S U/NSU字段控制的安全性和特权级别决定。	读写	0x1
5	<b>CORE1:</b> 若为1，DMA可由core 1访问，访问权限受本寄存器中SP/NSP/ SU/NSU允许的安全性与特权级别限制。	读写	0x1
4	<b>CORE0:</b> 若为1，DMA可由core 0访问，访问权限受本寄存器中SP/NSP/ SU/NSU允许的安全性与特权级别限制。	读写	0x1
3	<b>SP:</b> 若为1，DMA可从安全且具特权的上下文访问。	读写	0x1
2	<b>SU:</b> 若为1且SP同时被设置，DMA可从安全且非特权的上下文访问。	读写	0x1
1	<b>NSP:</b> 若为1，DMA可从非安全且具特权的上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1且NSP同时被设置，DMA可从非安全且非特权的上下文访问 。 仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：USBCTRL 寄存器

偏移量：0x48

### 描述

控制调试器、DMA、core 0及core 1的USBCTRL访问权限，以及其对应的安全性与特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表930. USBCTRL  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若设为1，则USBCTRL可由调试器访问，访问权限受本寄存器中SP/N SP/SU/NSU的安全与特权级别限制。	读写	0x1
6	<b>DMA:</b> 若设为1，则USBCTRL可由DMA访问，访问权限受本寄存器中SP/NSP/ SU/NSU的安全与特权级别限制。	读写	0x1
5	<b>CORE1:</b> 若设为1，则USBCTRL可由核心1访问，访问权限受本寄存器中SP/N SP/SU/NSU的安全与特权级别限制。	读写	0x1
4	<b>CORE0:</b> 若设为1，则USBCTRL可由核心0访问，访问权限受本寄存器中SP/N SP/SU/NSU的安全与特权级别限制。	读写	0x1
3	<b>SP:</b> 若设为1，则USBCTRL可从安全且具特权的上下文访问。	读写	0x1
2	<b>SU:</b> 若设为1且SP亦为1，则USBCTRL可从安全且无特权的上下文访问 。 。	读写	0x1
1	<b>NSP:</b> 若为1，则可从非安全的特权上下文访问 USBCTRL。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 若为1, 且 NSP 也被设置, 则可从非安全的非特权上下文访问 USBCTR L。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: PIO0 寄存器

偏移: 0x4c

### 说明

控制调试器、DMA、核心0和核心1是否能访问 PIO0, 以及其可访问的安全性与权限级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表931. PIO0 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, 可由调试器访问 PIO0, 访问权限由本寄存器中的 SP/NSP/SU /NSU 字段决定。	读写	0x1
6	<b>DMA</b> : 若为1, 可由 DMA 访问 PIO0, 访问权限由本寄存器中的 SP/NSP /SU/NSU 字段决定。	读写	0x1
5	<b>CORE1</b> : 若为1, 可由核心1访问 PIO0, 访问权限由本寄存器中的 SP/N SP/SU/NSU 字段决定。	读写	0x1
4	<b>CORE0</b> : 若为1, 则PIO0可被核心0访问, 访问权限由本寄存器中SP/NS P/SU/NSU的安全及特权级别决定。	读写	0x1
3	<b>SP</b> : 若为1, 则PIO0可从安全且具有特权的上下文访问。	读写	0x1
2	<b>SU</b> : 若为1且SP同时被设置, 则PIO0可从安全且无特权的上下文访问。	读写	0x1
1	<b>NSP</b> : 若为1, 则PIO0可从非安全且具有特权的上下文访问。	读写	0x0
0	<b>NSU</b> : 若为1且NSP同时被设置, 则PIO0可从非安全且无特权的上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: PIO1寄存器

偏移: 0x50

### 说明

控制调试器、DMA、核心0和核心1对PIO1的访问权限及其安全/特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表932. PIO1 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1，则PIO1可被调试器访问，访问权限由本寄存器中SP/NSP/SU/NSU的安全及特权级别决定。	读写	0x1
6	<b>DMA</b> : 若值为1，PIO1 可由 DMA 访问，访问级别受限于本寄存器中 SP/NSP/SU/NSU 设定的安全/权限等级。	读写	0x1
5	<b>CORE1</b> : 若值为1，PIO1 可由核心1访问，访问级别受限于本寄存器中 SP/NSP/SU/NSU 设定的安全/权限等级。	读写	0x1
4	<b>CORE0</b> : 若值为1，PIO1 可由核心0访问，访问级别受限于本寄存器中 SP/NSP/SU/NSU 设定的安全/权限等级。	读写	0x1
3	<b>SP</b> : 若值为1，PIO1 可从安全特权上下文进行访问。	读写	0x1
2	<b>SU</b> : 若值为1，且 SP 同时被设置，PIO1 可从安全非特权上下文进行访问。	读写	0x1
1	<b>NSP</b> : 若值为1，PIO1 可从非安全特权上下文进行访问。	读写	0x0
0	<b>NSU</b> : 若值为1，且 NSP 同时被设置，PIO1 可从非安全非特权上下文进行访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：PIO2 寄存器

偏移量: 0x54

### 描述

控制调试器、DMA、核心0与核心1是否能够访问PIO2，以及它们所能使用的安全/权限级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表933. PIO2 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1，调试器可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问PIO2。	读写	0x1
6	<b>DMA</b> : 若为1，DMA可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问PIO2。	读写	0x1
5	<b>CORE1</b> : 若为1，核心1可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问PIO2。	读写	0x1
4	<b>CORE0</b> : 若为1，核心0可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问PIO2。	读写	0x1
3	<b>SP</b> : 若为1，PIO2可从安全且特权的上下文访问。	读写	0x1
2	<b>SU</b> : 若为1，且SP同时被设置，则可从安全非特权上下文访问PIO2。	读写	0x1
1	<b>NSP</b> : 若为1，则可从非安全特权上下文访问PIO2。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 若为1, 且NSP同时被设置, 则可从非安全非特权上下文访问PIO2。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: CORESIGHT\_TRACE 寄存器

偏移: 0x58

### 描述

控制调试器、DMA、核心0和核心1是否可访问CORESIGHT\_TRACE, 以及其访问的安全性及权限级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表934。  
CORESIGHT\_TRACE  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, CORESIGHT_TRACE可被调试器访问, 访问级别须符合本寄存器中SP/NSP/SU/NSU所允许的安全性及权限。	读写	0x1
6	<b>DMA</b> : 若为1, CORESIGHT_TRACE可被DMA访问, 访问级别须符合本寄存器中SP/NSP/SU/NSU所允许的安全性及权限。	读写	0x0
5	<b>CORE1</b> : 若为1, CORESIGHT_TRACE 可由核心1访问, 访问权限受此寄存器中 SP/NSP/SU/NSU 所允许的安全/特权级别限制。	读写	0x1
4	<b>CORE0</b> : 若为1, CORESIGHT_TRACE 可由核心0访问, 访问权限受此寄存器中 SP/NSP/SU/NSU 所允许的安全/特权级别限制。	读写	0x1
3	<b>SP</b> : 若为1, CORESIGHT_TRACE 可从安全且特权上下文访问。	读写	0x1
2	<b>SU</b> : 若为1, 且 SP 亦已设置, 则 CORESIGHT_TRACE 可从安全且无特权上下文访问。	读写	0x0
1	<b>NSP</b> : 若为1, CORESIGHT_TRACE 可从非安全且特权上下文访问。	读写	0x0
0	<b>NSU</b> : 若为1, 且 NSP 亦已设置, 则 CORESIGHT_TRACE 可从非安全且无特权上下文访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: CORESIGHT\_PERIPH 寄存器

偏移: 0x5c

### 描述

控制调试器、DMA、核心0及核心1是否可访问 CORESIGHT\_PERIPH 以及其访问的安全/特权级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 935。  
CORESIGHT\_PERIPH  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若值为1，调试器可依照本寄存器中SP/NSP/SU/NSU所允许的安全性及权限级别访问CORESIGHT_PERIPH。	读写	0x1
6	<b>DMA:</b> 若值为1，DMA可依照本寄存器中SP/NSP/SU/NSU所允许的安全性及权限级别访问CORESIGHT_PERIPH。	读写	0x0
5	<b>CORE1:</b> 若值为1，核1可依照本寄存器中SP/NSP/SU/NSU所允许的安全性及权限级别访问CORESIGHT_PERIPH。	读写	0x1
4	<b>CORE0:</b> 若值为1，核0可依照本寄存器中SP/NSP/SU/NSU所允许的安全性及权限级别访问CORESIGHT_PERIPH。	读写	0x1
3	<b>SP:</b> 若值为1，CORESIGHT_PERIPH可从安全且特权的上下文访问。	读写	0x1
2	<b>SU:</b> 若值为1，且SP亦被设置，CORESIGHT_PERIPH可从安全且非特权的上下文访问。	读写	0x0
1	<b>NSP:</b> 若为1，CORESIGHT_PERIPH可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1且NSP亦设置，CORESIGHT_PERIPH可从非安全非特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：SYSINFO 寄存器

偏移: 0x60

### 描述

控制调试器、DMA、核心0及核心1是否能够访问SYSINFO，以及访问时所需的安全性和特权级别。

默认情况下，访问权限完全开放。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表 936. SYSINFO  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1，调试器可根据本寄存器中SP/NSP/SU/NSU字段允许的安全性和特权级别访问SYSINFO。	读写	0x1
6	<b>DMA:</b> 若为1，DMA可根据本寄存器中SP/NSP/SU/NSU字段允许的安全性和特权级别访问SYSINFO。	读写	0x1
5	<b>CORE1:</b> 若为1，核心1可根据本寄存器中SP/NSP/SU/NSU字段允许的安全性和特权级别访问SYSINFO。	读写	0x1
4	<b>CORE0:</b> 若为1，SYSINFO可被核心0依据本寄存器内SP/NSP/SU/NSU所允许的安全／特权级别访问。	读写	0x1
3	<b>SP:</b> 若为1，SYSINFO 可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若为1，且SP同时被设置，SYSINFO可从安全非特权上下文访问。	读写	0x1

位	描述	类型	复位
1	<b>NSP</b> : 若为1, SYSINFO 可从非安全特权上下文访问。	读写	0x1
0	<b>NSU</b> : 若为1, 且NSP同时被设置, SYSINFO可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x1

## ACCESSCTRL: RESETS 寄存器

偏移量: 0x64

### 说明

控制调试器、DMA、核心0及核心1是否可以访问RESETS, 以及其访问所允许的安全／特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 937. RESETS 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, 调试器可在本寄存器中SP/NSP/SU/NSU所允许的安全／特权级别下访问RESETS。	读写	0x1
6	<b>DMA</b> : 如果设置为1, 则在本寄存器所允许的SP/NSP/SU/NSU安全/特权级别范围内, DMA可以访问RESETS。	读写	0x1
5	<b>CORE1</b> : 如果设置为1, 则在本寄存器所允许的SP/NSP/SU/NSU安全/特权级别范围内, 核1可以访问RESETS。	读写	0x1
4	<b>CORE0</b> : 如果设置为1, 则在本寄存器所允许的SP/NSP/SU/NSU安全/特权级别范围内, 核0可以访问RESETS。	读写	0x1
3	<b>SP</b> : 如果设置为1, 则可以从安全特权上下文访问RESETS。	读写	0x1
2	<b>SU</b> : 如果设置为1, 且SP同时被设置, 则可以从安全非特权上下文访问RESETS。	读写	0x1
1	<b>NSP</b> : 如果设置为1, 则可以从非安全特权上下文访问RESETS。	读写	0x0
0	<b>NSU</b> : 如果设置为1, 且NSP同时被设置, 则可以从非安全非特权上下文访问RESETS。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: IO\_BANK0 寄存器

偏移: 0x68

### 描述

控制调试器、DMA、核0和核1是否可以访问 IO\_BANK0 以及其访问所处的安全/权限级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 938. IO\_BANK0 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1, IO_BANK0 可被调试器访问, 访问权限按照本寄存器中 SP / NSP/SU/NSU 指定的安全/权限级别执行。	读写	0x1
6	<b>DMA</b> : 如果为1, IO_BANK0 可被 DMA 访问, 访问权限按照本寄存器中 SP/N SP/SU/NSU 指定的安全/权限级别执行。	读写	0x1
5	<b>CORE1</b> : 如果为1, IO_BANK0 可被核1访问, 访问权限按照本寄存器中 SP/N SP/SU/NSU 指定的安全/权限级别执行。	读写	0x1
4	<b>CORE0</b> : 如果为1, IO_BANK0 可被核0访问, 访问权限按照本寄存器中 SP/N SP/SU/NSU 指定的安全/权限级别执行。	读写	0x1
3	<b>SP</b> : 如果为1, 则可从安全且特权的上下文访问IO_BANK0。	读写	0x1
2	<b>SU</b> : 如果为1且SP同样被设置, 则可从安全且非特权的上下文访问IO_BANK0。	读写	0x1
1	<b>NSP</b> : 如果为1, 则可从非安全且特权的上下文访问IO_BANK0。	读写	0x0
0	<b>NSU</b> : 如果为1且NSP同样被设置, 则可从非安全且非特权的上下文访问IO_BANK0。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：IO\_BANK1寄存器

偏移: 0x6c

### 描述

控制调试器、DMA、核心0和核心1对IO\_BANK1的访问权限及其安全／特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 939. IO\_BANK1 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1, 调试器可根据本寄存器中SP / NSP / SU / NSU所允许的安全／特权级别访问IO_BANK1。	读写	0x1
6	<b>DMA</b> : 如果为1, DMA可根据本寄存器中SP / NSP / SU / NSU所允许的安全／特权级别访问IO_BANK1。	读写	0x1
5	<b>CORE1</b> : 若为1, 则IO_BANK1可由core 1访问, 访问权限受本寄存器中SP/NS P/SU/NSU的安全与特权级别限制。	读写	0x1
4	<b>CORE0</b> : 若为1, 则IO_BANK1可由core 0访问, 访问权限受本寄存器中SP/NS P/SU/NSU的安全与特权级别限制。	读写	0x1
3	<b>SP</b> : 若为1, IO_BANK1可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 若为1且SP同时被设置, IO_BANK1可从安全非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 若为1, IO_BANK1可从非安全特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU:</b> 若为1且NSP同时被设置, IO_BANK1可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: PADS\_BANK0寄存器

偏移: 0x70

### 描述

控制调试器、DMA、核心0与核心1是否可以访问PADS\_BANK0, 以及它们可在本寄存器中由SP/NSP/SU/NSU定义的何种安全/特权级别下访问。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表940。  
PADS\_BANK0寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可按照本寄存器中SP/NSP/SU/NSU许可的安全/特权级别访问PADS_BANK0。	读写	0x1
6	<b>DMA:</b> 若为1, DMA可按照本寄存器中SP/NSP/SU/NSU许可的安全/特权级别访问PADS_BANK0。	读写	0x1
5	<b>CORE1:</b> 若为1, 核心1可按照本寄存器中SP/NSP/SU/NSU许可的安全/特权级别访问PADS_BANK0。	读写	0x1
4	<b>CORE0:</b> 若为1, 核心0可按照本寄存器中SP/NSP/SU/NSU许可的安全/特权级别访问PADS_BANK0。	读写	0x1
3	<b>SP:</b> 如果为1, PADS_BANK0可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1, 且SP同时设置, PADS_BANK0可从安全非特权上下文访问。 ◦	读写	0x1
1	<b>NSP:</b> 如果为1, PADS_BANK0可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 如果为1, 且NSP同时设置, PADS_BANK0可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: PADS\_QSPI寄存器

偏移: 0x74

### 描述

控制调试器、DMA、核0与核1对PADS\_QSPI的访问权限及其安全／特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表941. PADS\_QSPI  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可访问PADS_QSPI, 访问安全／特权级别由本寄存器中SP／NSP／SU／NSU字段决定。	读写	0x1
6	<b>DMA:</b> 若为1, DMA可访问PADS_QSPI, 访问安全／特权级别由本寄存器中SP／NSP／SU／NSU字段决定。	读写	0x1
5	<b>CORE1:</b> 若为1, 则PADS_QSPI可由核心1访问, 访问权限遵循此寄存器中SP/N SP/SU/NSU所定义的安全及特权级别。	读写	0x1
4	<b>CORE0:</b> 若为1, 则PADS_QSPI可由核心0访问, 访问权限遵循此寄存器中SP/N SP/SU/NSU所定义的安全及特权级别。	读写	0x1
3	<b>SP:</b> 若为1, 则PADS_QSPI可从安全的特权上下文访问。	读写	0x1
2	<b>SU:</b> 若为1, 且SP已设置, 则PADS_QSPI可从安全的非特权上下文访问 。	读写	0x1
1	<b>NSP:</b> 若为1, 则PADS_QSPI可从非安全的特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1, 且NSP已设置, 则PADS_QSPI可从非安全的非特权上下文 访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: BUSCTRL寄存器

偏移量: 0x78

### 描述

控制调试器、DMA、核心0和核心1是否可以访问BUSCTRL, 以及它们在何种安全和特权级别下可执行访问。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表942. BUSCTRL  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若值为1, 调试器可在本寄存器中SP/NSP/SU/NSU允许的安全/权限 等级下访问BUSCTRL。	读写	0x1
6	<b>DMA:</b> 若值为1, DMA可在本寄存器中SP/NSP/SU/NSU允许的安全/权限等级 下访问BUSCTRL。	读写	0x1
5	<b>CORE1:</b> 若值为1, 核心1可在本寄存器中SP/NSP/SU/NSP允许的安全/权限等 级下访问BUSCTRL。	读写	0x1
4	<b>CORE0:</b> 若值为1, 核心0可在本寄存器中SP/NSP/SU/NSP允许的安全/权限等 级下访问BUSCTRL。	读写	0x1
3	<b>SP:</b> 若值为1, 可从安全且特权上下文访问BUSCTRL。	读写	0x1
2	<b>SU:</b> 若值为1, 且SP也被设置, 则可从安全且非特权上下文访问BUSC TRL。	读写	0x1
1	<b>NSP:</b> 若为1, BUSCTRL 可从非安全特权上下文进行访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 若为1, 且NSP亦已设置, BUSCTRL可从非安全无特权上下文进行访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: ADC寄存器

偏移: 0x7c

### 说明

控制调试器、DMA、核心0及核心1是否能够访问ADC，以及其对应的安全/特权访问级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表 943. ADC  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, 根据本寄存器中SP/NSP/SU/NSU字段所允许的安全及特权级别, 调试器可访问ADC。	读写	0x1
6	<b>DMA</b> : 若为1, 根据本寄存器中SP/NSP/SU/NSU字段所允许的安全及特权级别, DMA可访问ADC。	读写	0x1
5	<b>CORE1</b> : 若为1, 根据本寄存器中SP/NSP/SU/NSU字段所允许的安全及特权级别, 核心1可访问ADC。	读写	0x1
4	<b>CORE0</b> : 若为1, ADC 可由核心0访问, 访问权限受本寄存器中 SP/NSP /SU/NSU 所允许的安全性及特权级别限制。	读写	0x1
3	<b>SP</b> : 若为1, ADC 可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 若为1且SP亦被设置, ADC 可从安全非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 若为1, ADC 可从非安全特权上下文访问。	读写	0x0
0	<b>NSU</b> : 若为1且NSP亦被设置, ADC 可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: HSTX 寄存器

偏移量: 0x80

### 描述

控制调试器、DMA、核心0及核心1是否能够访问HSTX，以及其访问的安全性与特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表944. HSTX  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可访问HSTX, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全性及特权级别限制。	读写	0x1
6	<b>DMA:</b> 如果为1, HSTX可被DMA访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
5	<b>CORE1:</b> 如果为1, HSTX可被核心1访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
4	<b>CORE0:</b> 如果为1, HSTX可被核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
3	<b>SP:</b> 如果为1, HSTX可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1, 且SP亦被设置, HSTX可从安全非特权上下文访问。	读写	0x1
1	<b>NSP:</b> 如果为1, HSTX可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 如果为1, 且NSP亦被设置, HSTX可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: I2C0寄存器

偏移: 0x84

### 描述

控制调试器、DMA、核心0和核心1是否可访问I2C0, 以及它们可在何种安全/权限级别下访问。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表945. I2C0  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若该位为1, 调试器可按照本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问I2C0。	读写	0x1
6	<b>DMA:</b> 若该位为1, DMA可按照本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问I2C0。	读写	0x1
5	<b>CORE1:</b> 若该位为1, 核心1可按照本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问I2C0。	读写	0x1
4	<b>CORE0:</b> 若该位为1, 核心0可按照本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问I2C0。	读写	0x1
3	<b>SP:</b> 若为1, 则I2C0可从安全特权环境访问。	读写	0x1
2	<b>SU:</b> 若为1, 且SP也被设置, 则I2C0可从安全非特权环境访问。	读写	0x1
1	<b>NSP:</b> 若为1, 则I2C0可从非安全特权环境访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 若为1, 且NSP也被设置, 则I2C0可从非安全非特权环境访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: I2C1寄存器

偏移量: 0x88

### 描述

控制调试器、DMA、核心0和核心1对I2C1的访问权限及其安全/特权等级。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表946. I2C1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, 调试器可按照本寄存器中SP/NSP/SU/NSU位允许的安全/特权等级访问I2C1。	读写	0x1
6	<b>DMA</b> : 如果为1, 则在本寄存器中SP/NSP/SU/NSU允许的安全性和特权级别下, DMA可访问I2C1。	读写	0x1
5	<b>CORE1</b> : 如果为1, 则在本寄存器中SP/NSP/SU/NSU允许的安全性和特权级别下, 核心1可访问I2C1。	读写	0x1
4	<b>CORE0</b> : 如果为1, 则在本寄存器中SP/NSP/SU/NSU允许的安全性和特权级别下, 核心0可访问I2C1。	读写	0x1
3	<b>SP</b> : 如果为1, 则可从安全的特权上下文访问I2C1。	读写	0x1
2	<b>SU</b> : 如果为1且SP同时设置, 则可从安全的非特权上下文访问I2C1。	读写	0x1
1	<b>NSP</b> : 如果为1, 则可从非安全的特权上下文访问I2C1。	读写	0x0
0	<b>NSU</b> : 如果为1且NSP同时设置, 则可从非安全的非特权上下文访问I2C1。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: PWM 寄存器

偏移: 0x8c

### 描述

控制调试器、DMA、核心0及核心1是否可访问PWM, 以及它们的访问安全/权限级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 947. PWM 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 当值为 1 时, PWM 可被调试器访问, 访问权限依本寄存器中 SP/NSP/SU/NSU 的安全/权限级别而定。	读写	0x1
6	<b>DMA</b> : 当值为 1 时, PWM 可被 DMA 访问, 访问权限依本寄存器中 SP/NSP/SU/NSU 的安全/权限级别而定。	读写	0x1
5	<b>CORE1</b> : 当值为 1 时, PWM 可被核心 1 访问, 访问权限依本寄存器中 SP/NSP/SU/NSU 的安全/权限级别而定。	读写	0x1
4	<b>CORE0</b> : 当值为 1 时, PWM 可被核心 0 访问, 访问权限依本寄存器中 SP/NSP/SU/NSU 的安全/权限级别而定。	读写	0x1
3	<b>SP</b> : 当值为 1 时, PWM 可由安全且具特权的上下文访问。	读写	0x1
2	<b>SU</b> : 若值为 1, 且 SP 亦被设置, 则 PWM 可从安全无特权环境访问。	读写	0x1
1	<b>NSP</b> : 若值为 1, PWM 可从非安全特权环境访问。	读写	0x0
0	<b>NSU</b> : 若值为 1, 且 NSP 亦被设置, 则 PWM 可从非安全无特权环境访问。  仅当 NSP 位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：SPI0 寄存器

偏移量: 0x90

### 描述

控制调试器、DMA、核心0及核心1是否可访问SPI0, 以及其可访问的安全/特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 948. SPI0 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若值为 1, 则调试器可按本寄存器中 SP/NSP/SU/NSU 允许的安全/特权级别访问 SPI0。	读写	0x1
6	<b>DMA</b> : 若值为 1, 则 DMA 可按本寄存器中 SP/NSP/SU/NSU 允许的安全/特权级别访问 SPI0。	读写	0x1
5	<b>CORE1</b> : 若值为 1, 则核心 1 可按本寄存器中 SP/NSP/SU/NSU 允许的安全/特权级别访问 SPI0。	读写	0x1
4	<b>CORE0</b> : 如果为 1, 则 SPI0 可由核心 0 访问, 访问权限受本寄存器中 SP/NSP/SU/NSU 允许的安全/特权级别限制。	读写	0x1
3	<b>SP</b> : 如果为 1, 则 SPI0 可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 如果为 1 且 SP 也已设置, 则 SPI0 可从安全非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 如果为 1, 则 SPI0 可从非安全特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 如果为1且NSP也已设置，则SPI0可从非安全非特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL： SPI1 寄存器

偏移量: 0x94

### 描述

控制调试器、DMA、核心0及核心1对SPI1的访问权限及其所允许的安全/特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表949. SPI1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1，则调试器可访问SPI1，访问权限受本寄存器中SP/NSP/SU/ NSU允许的安全/特权级别限制。	读写	0x1
6	<b>DMA</b> : 若值为1，则SPI1可由DMA访问，访问权限受本寄存器中SP/NSP/ /SU/NSU所允许的安全/特权级别限制。	读写	0x1
5	<b>CORE1</b> : 若值为1，则SPI1可由核1访问，访问权限受本寄存器中SP/NSP/ /SU/NSU所允许的安全/特权级别限制。	读写	0x1
4	<b>CORE0</b> : 若值为1，则SPI1可由核0访问，访问权限受本寄存器中SP/NSP/ /SU/NSU所允许的安全/特权级别限制。	读写	0x1
3	<b>SP</b> : 若值为1，则SPI1可从安全的特权上下文访问。	读写	0x1
2	<b>SU</b> : 若值为1，且SP亦被设置，则SPI1可从安全的非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 若值为1，则SPI1可从非安全的特权上下文访问。	读写	0x0
0	<b>NSU</b> : 若值为1，且NSP亦被设置，则SPI1可从非安全的非特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL： TIMERO 寄存器

偏移: 0x98

### 描述

控制调试器、DMA、核心0和核心1是否可访问 TIMERO 及其允许的安全/权限级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表 950. TIMER0 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1，则调试器可依据本寄存器中 SP/NSP/SU/NSU 允许的安全/权限级别访问 TIMER0。	读写	0x1
6	<b>DMA:</b> 若为1，则 DMA 可依据本寄存器中 SP/NSP/SU/NSU 允许的安全/权限级别访问 TIMER0。	读写	0x1
5	<b>CORE1:</b> 若为1，则核心1可依据本寄存器中 SP/NSP/SU/NSU 允许的安全/权限级别访问 TIMER0。	读写	0x1
4	<b>CORE0:</b> 若为1，则核心0可依据本寄存器中 SP/NSP/SU/NSU 允许的安全/权限级别访问 TIMER0。	读写	0x1
3	<b>SP:</b> 若为1，TIMER0可从安全且有特权的上下文访问。	读写	0x1
2	<b>SU:</b> 若为1，且SP已设置，TIMER0可从安全的无特权上下文访问。	读写	0x1
1	<b>NSP:</b> 若为1，TIMER0可从非安全且有特权的上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1，且NSP已设置，TIMER0可从非安全的无特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：TIMER1寄存器

偏移: 0x9c

### 描述

控制调试器、DMA、核心0及核心1是否可访问TIMER1，以及其访问的安全性/特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表 951. TIMER1 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1，调试器可按本寄存器中SP/NSP/SU/NSU所限的安全性/特权级别访问TIMER1。	读写	0x1
6	<b>DMA:</b> 若为1，DMA可按本寄存器中SP/NSP/SU/NSU所限的安全性/特权级别访问TIMER1。	读写	0x1
5	<b>CORE1:</b> 如果为1，TIMER1 可由核心1访问，前提是本寄存器内 SP/NSP/SU/NSU 允许的安全/权限级别。	读写	0x1
4	<b>CORE0:</b> 如果为1，TIMER1 可由核心0访问，前提是本寄存器内 SP/NSP/SU/NSU 允许的安全/权限级别。	读写	0x1
3	<b>SP:</b> 如果为1，TIMER1 可从安全且特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1，且同时设置了 SP，TIMER1 可从安全且非特权上下文访问。	读写	0x1
1	<b>NSP:</b> 如果为1，TIMER1 可从非安全且特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 如果为1, 且同时设置了 NSP, TIMER1 可从非安全且非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: UART0 寄存器

偏移: 0xa0

### 描述

控制调试器、DMA、核心0和核心1是否可以访问UART0 以及其安全/权限级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表952. UART0 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1, 则UART0可由调试器访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全及权限级别限制。	读写	0x1
6	<b>DMA</b> : 若为1, 则UART0可由DMA访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全及权限级别限制。	读写	0x1
5	<b>CORE1</b> : 若为1, 则UART0可由核心1访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全及权限级别限制。	读写	0x1
4	<b>CORE0</b> : 若为1, 则UART0可由核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU的安全及权限级别限制。	读写	0x1
3	<b>SP</b> : 若为1, 则UART0可从安全特权上下文访问。	读写	0x1
2	<b>SU</b> : 若为1且SP已设置, 则UART0可从安全非特权上下文访问。	读写	0x1
1	<b>NSP</b> : 若为1, UART0可由非安全特权上下文访问。	读写	0x0
0	<b>NSU</b> : 若为1, 且NSP同时设置, UART0可由非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: UART1寄存器

偏移: 0xa4

### 描述

控制调试器、DMA、核心0及核心1对UART1的访问权限及其安全性/特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表953. UART1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, UART1可由调试器访问, 访问权限遵循本寄存器中SP/NSP/SU/NSU的安全性及特权级别设定。	读写	0x1
6	<b>DMA:</b> 若为1, UART1可由DMA访问, 访问权限遵循本寄存器中SP/NSP/SU/NSU的安全性及特权级别设定。	读写	0x1
5	<b>CORE1:</b> 若为1, UART1可由核心1访问, 访问权限遵循本寄存器中SP/NSP/SU/NSU的安全性及特权级别设定。	读写	0x1
4	<b>CORE0:</b> 若为1, 则UART1可由核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU定义的安全／特权级别限制。	读写	0x1
3	<b>SP:</b> 若为1, 则UART1可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若为1且SP已设置, 则UART1可从安全无特权上下文访问。	读写	0x1
1	<b>NSP:</b> 若为1, 则UART1可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1且NSP已设置, 则UART1可从非安全无特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：OTP寄存器

偏移: 0xa8

### 描述

控制调试器、DMA、核心0及核心1是否可访问OTP及其访问的安全／特权级别。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表954. OTP  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 则调试器可访问OTP, 访问权限受本寄存器中SP/NSP/SU/NSU定义的安全／特权级别限制。	读写	0x1
6	<b>DMA:</b> 若值为1, 则OTP可由DMA访问, 访问权限依据本寄存器中SP/NSP/SU/NSU所允许的安全及特权级别。	读写	0x1
5	<b>CORE1:</b> 若值为1, 则OTP可由核心1访问, 访问权限依据本寄存器中SP/NSP/SU/NSU所允许的安全及特权级别。	读写	0x1
4	<b>CORE0:</b> 若值为1, 则OTP可由核心0访问, 访问权限依据本寄存器中SP/NSP/SU/NSU所允许的安全及特权级别。	读写	0x1
3	<b>SP:</b> 若值为1, 则OTP可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若值为1且SP同时设置, 则OTP可从安全非特权上下文访问。	读写	0x1
1	<b>NSP:</b> 若值为1, 则OTP可从非安全特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU:</b> 若值为1且NSP同时设置，则OTP可从非安全非特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：TBMAN寄存器

偏移: 0xac

### 描述

控制调试器、DMA、核心0和核心1是否能够访问TBMAN，以及它们可以在何种安全/特权级别下进行访问。

默认允许任何主设备进行安全访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表 955. TBMAN 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1，调试器可按本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别访问TBMAN。	读写	0x1
6	<b>DMA:</b> 若为1，DMA可按本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别访问TBMAN。	读写	0x1
5	<b>CORE1:</b> 若为1，核心1可按本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别访问TBMAN。	读写	0x1
4	<b>CORE0:</b> 若为1，核心0可按本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别访问TBMAN。	读写	0x1
3	<b>SP:</b> 若为1，TBMAN可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1，且SP被设置，则可从安全非特权上下文访问TBMAN ◦	读写	0x1
1	<b>NSP:</b> 如果为1，则可从非安全特权上下文访问TBMAN。	读写	0x0
0	<b>NSU:</b> 如果为1，且NSP被设置，则可从非安全非特权上下文访问TBMAN。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：POWMAN寄存器

偏移: 0xb0

### 说明

控制调试器、DMA、核0和核1是否可以访问POWMAN，以及允许它的安全/特权级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表956. POWMAN  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1，则调试器可在本寄存器SP/NSP/SU/NSU允许的安全/特权级别下访问POWMAN。	读写	0x1
6	<b>DMA</b> : 如果为1，则DMA可在本寄存器SP/NSP/SU/NSU允许的安全/特权级别下访问POWMAN。	读写	0x0
5	<b>CORE1</b> : 如果为1，POWMAN可由核心1访问，访问权限受本寄存器中SP、NSP、SU、NSU项所规定的安全及特权等级限制。	读写	0x1
4	<b>CORE0</b> : 如果为1，POWMAN可由核心0访问，访问权限受本寄存器中SP、NSP、SU、NSU项所规定的安全及特权等级限制。	读写	0x1
3	<b>SP</b> : 如果为1，POWMAN可从安全且特权的上下文访问。	读写	0x1
2	<b>SU</b> : 如果为1且同时设置了SP，POWMAN可从安全且非特权的上下文访问。	读写	0x0
1	<b>NSP</b> : 如果为1，POWMAN可从非安全且特权的上下文访问。	读写	0x0
0	<b>NSU</b> : 如果为1且同时设置了NSP，POWMAN可从非安全且非特权的上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：真随机数发生器（TRNG）寄存器

偏移: 0xb4

### 描述

控制调试器、DMA、核心0及核心1是否可以访问TRNG，以及可访问的安全与特权等级。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表957. TRNG  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 若为1，则调试器可在此寄存器所允许的SP/NSP/SU/NSU安全与特权级别下访问TRNG。	读写	0x1
6	<b>DMA</b> : 若为1，则DMA可在此寄存器所允许的SP/NSP/SU/NSU安全与特权级别下访问TRNG。	读写	0x0
5	<b>CORE1</b> : 若为1，则核心1可在此寄存器所允许的SP/NSP/SU/NSU安全与特权级别下访问TRNG。	读写	0x1
4	<b>CORE0</b> : 若为1，则核心0可在此寄存器所允许的SP/NSP/SU/NSU安全与特权级别下访问TRNG。	读写	0x1
3	<b>SP</b> : 若为1，则可从安全特权上下文访问TRNG。	读写	0x1
2	<b>SU</b> : 若为1，且SP同时被设置，则可从安全非特权上下文访问TRNG。	读写	0x0
1	<b>NSP</b> : 若为1，TRNG可从非安全特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU:</b> 若为1, 且NSP同时设置, TRNG可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：SHA256寄存器

偏移量: 0xb8

### 描述

控制调试器、DMA、核心0及核心1是否可访问SHA256, 以及它们可在哪些安全和特权级别下访问。

默认仅允许安全特权访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表958. SHA256  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可在本寄存器中SP/NSP/SU/NSU所允许的安全和特权级别下访问SHA256。	读写	0x1
6	<b>DMA:</b> 若为1, DMA可在本寄存器中SP/NSP/SU/NSU所允许的安全和特权级别下访问SHA256。	读写	0x1
5	<b>CORE1:</b> 若为1, 核心1可在本寄存器中SP/NSP/SU/NSU所允许的安全和特权级别下访问SHA256。	读写	0x1
4	<b>CORE0:</b> 若为1, SHA256可被核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU安全/特权级别的限制。	读写	0x1
3	<b>SP:</b> 若为1, SHA256 可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若为1且SP同时被置位, SHA256可从安全非特权上下文访问。	读写	0x0
1	<b>NSP:</b> 若为1, SHA256 可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1且NSP同时被置位, SHA256可从非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：SYSCFG寄存器

偏移量: 0xbc

### 描述

控制调试器、DMA、核心0和核心1是否能访问SYSCFG及其访问所允许的安全/特权级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表959. SYSCFG 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可访问SYSCFG, 访问权限受本寄存器中SP/NSP/SU/NSU安全/特权级别的限制。	读写	0x1
6	<b>DMA:</b> 若为1, SYSCFG可由DMA访问, 其权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x0
5	<b>CORE1:</b> 若为1, SYSCFG可由core 1访问, 其权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
4	<b>CORE0:</b> 若为1, SYSCFG可由core 0访问, 其权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
3	<b>SP:</b> 若为1, SYSCFG可由安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若为1且SP同时设置, SYSCFG可由安全非特权上下文访问。	读写	0x0
1	<b>NSP:</b> 若为1, SYSCFG可由非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1且NSP同时设置, SYSCFG可由非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: CLOCKS寄存器

偏移: 0xc0

### 描述

控制调试器、DMA、核心0和核心1是否能够访问CLOCKS, 以及它们在何种安全/权限级别下可进行访问。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表960. CLOCKS 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, 调试器可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问CLOCKS。	读写	0x1
6	<b>DMA:</b> 若为1, DMA可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问CLOCKS。	读写	0x0
5	<b>CORE1:</b> 若为1, 核心1可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问CLOCKS。	读写	0x1
4	<b>CORE0:</b> 若为1, 核心0可按本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别访问CLOCKS。	读写	0x1
3	<b>SP:</b> 若为1, 可从安全且具特权的上下文访问CLOCKS。	读写	0x1
2	<b>SU:</b> 如果为1, 且SP也被设置, 则可从安全非特权上下文访问CLOCKS。	读写	0x0
1	<b>NSP:</b> 如果为1, 则可从非安全特权上下文访问CLOCKS。	读写	0x0

位	描述	类型	复位
0	<b>NSU:</b> 如果为1, 且NSP也被设置, 则可从非安全非特权上下文访问CLOCKS。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: XOSC 寄存器

偏移: 0xc4

### 说明

控制调试器、DMA、核心0和核心1是否可以访问XOSC, 以及它们可在何种安全性/权限级别下访问。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表961. XOSC  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 如果为1, 调试器可根据本寄存器中SP、NSP、SU和NSU允许的安全性与权限级别访问XOSC。	读写	0x1
6	<b>DMA:</b> 如果为1, DMA可根据本寄存器中SP、NSP、SU和NSU允许的安全性与权限级别访问XOSC。	读写	0x0
5	<b>CORE1:</b> 若设置为1, 则XOSC可由核心1按本寄存器中SP/NSP/SU/NSU允许的安全/权限级别访问。	读写	0x1
4	<b>CORE0:</b> 若设置为1, 则XOSC可由核心0按本寄存器中SP/NSP/SU/NSU允许的安全/权限级别访问。	读写	0x1
3	<b>SP:</b> 若设置为1, XOSC可从安全、特权上下文访问。	读写	0x1
2	<b>SU:</b> 若设置为1, 且SP也被设置, XOSC可从安全、非特权上下文访问。	读写	0x0
1	<b>NSP:</b> 若设置为1, XOSC可从非安全、特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若设置为1, 且NSP也被设置, XOSC可从非安全、非特权上下文访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: ROSC寄存器

偏移: 0xc8

### 描述

控制调试器、DMA、核心0和核心1是否可以访问ROSC, 以及其访问的安全/权限级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 962. ROSC 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1，则可在本寄存器中SP/NSP/SU/NSU允许的安全与特权级别下，通过调试器访问ROSC。	读写	0x1
6	<b>DMA:</b> 若为1，则可在本寄存器中SP/NSP/SU/NSU允许的安全与特权级别下，通过DMA访问ROSC。	读写	0x0
5	<b>CORE1:</b> 若为1，则可在本寄存器中SP/NSP/SU/NSU允许的安全与特权级别下，由核1访问ROSC。	读写	0x1
4	<b>CORE0:</b> 若为1，则可在本寄存器中SP/NSP/SU/NSU允许的安全与特权级别下，由核0访问ROSC。	读写	0x1
3	<b>SP:</b> 若为1，则可从安全且具有特权的上下文访问ROSC。	读写	0x1
2	<b>SU:</b> 若为1，且SP同时被设置，则可从安全且非特权的上下文访问ROSC。	读写	0x0
1	<b>NSP:</b> 若值为1，则ROSC可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 若值为1且NSP亦被设置，则ROSC可从非安全无特权上下文访问。  仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：PLL\_SYS寄存器

偏移量：0xcc

### 描述

控制调试器、DMA、核心0和核心1是否可访问PLL\_SYS，以及其可在哪些安全/特权级别下访问。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表963。PLL\_SYS 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若值为1，则PLL_SYS可由调试器访问，访问权限受本寄存器中SP/NSP/SU/NSU所定义的安全/特权级别限制。	读写	0x1
6	<b>DMA:</b> 若值为1，则PLL_SYS可由DMA访问，访问权限受本寄存器中SP/NSP/SU/NSU所定义的安全/特权级别限制。	读写	0x0
5	<b>CORE1:</b> 若值为1，则PLL_SYS可由核心1访问，访问权限受本寄存器中SP/NSP/SU/NSU所定义的安全/特权级别限制。	读写	0x1
4	<b>CORE0:</b> 如果为1，PLL_SYS可由核0访问，访问权限依照本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别。	读写	0x1
3	<b>SP:</b> 如果为1，PLL_SYS可从安全、特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1，且SP也被设置，PLL_SYS可从安全、非特权上下文访问。	读写	0x0
1	<b>NSP:</b> 如果为1，PLL_SYS可从非安全、特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU</b> : 如果为1, 且NSP也被设置, PLL_SYS可从非安全、非特权上下文访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: PLL\_USB寄存器

偏移量: 0xd0

### 说明

控制调试器、DMA、核0及核1对PLL\_USB的访问权限及其安全/特权级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表964. PLL\_USB  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG</b> : 如果为1, PLL_USB可由调试器访问, 访问权限依照本寄存器中SP/ NSP/SU/NSU所允许的安全/特权级别。	读写	0x1
6	<b>DMA</b> : 如果为1, PLL_USB可被DMA访问, 访问权限受本寄存器中SP/NSP/S U/NSU所允许的安全/权限级别限制。	读写	0x0
5	<b>CORE1</b> : 如果为1, PLL_USB可被核心1访问, 访问权限受本寄存器中SP/NS P/SU/NSU所允许的安全/权限级别限制。	读写	0x1
4	<b>CORE0</b> : 如果为1, PLL_USB可被核心0访问, 访问权限受本寄存器中SP/NS P/SU/NSU所允许的安全/权限级别限制。	读写	0x1
3	<b>SP</b> : 如果为1, PLL_USB可从安全、特权上下文访问。	读写	0x1
2	<b>SU</b> : 如果为1且SP也为1, PLL_USB可从安全、非特权上下文访问。	读写	0x0
1	<b>NSP</b> : 如果为1, PLL_USB可从非安全、特权上下文访问。	读写	0x0
0	<b>NSU</b> : 如果为1且NSP也为1, PLL_USB可从非安全、非特权上下文访问。 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: TICKS 寄存器

偏移: 0xd4

### 描述

控制调试器、DMA、第0核和第1核是否可以访问 TICKS, 以及它们能以何种安全/权限级别进行访问。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 965. TICKS 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, TICKS 可被调试器访问, 访问权限由本寄存器中 SP/NSP/SU/N SU 的安全/权限级别规定。	读写	0x1
6	<b>DMA:</b> 若为1, TICKS 可被 DMA 访问, 访问权限由本寄存器中 SP/NSP/SU/ NSU 的安全/权限级别规定。	读写	0x0
5	<b>CORE1:</b> 若为1, TICKS 可被第1核访问, 访问权限由本寄存器中 SP/NSP/ SU/NSU 的安全/权限级别规定。	读写	0x1
4	<b>CORE0:</b> 若为1, TICKS 可被第0核访问, 访问权限由本寄存器中 SP/NSP/ SU/NSU 的安全/权限级别规定。	读写	0x1
3	<b>SP:</b> 若值为1, TICKS可从安全且特权的上下文中访问。	读写	0x1
2	<b>SU:</b> 若值为1, 且SP亦被设置, TICKS可从安全且非特权的上下文 中访问。	读写	0x0
1	<b>NSP:</b> 若值为1, TICKS可从非安全且特权的上下文中访问。	读写	0x0
0	<b>NSU:</b> 若值为1, 且NSP亦被设置, TICKS可从非安全且非特权的上下文中 访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：看门狗寄存器

偏移: 0xd8

### 描述

控制调试器、DMA、核心0和核心1是否可访问看门狗, 以及访问时所处的安全/特权级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 966。看门狗寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若值为1, 调试器可按照本寄存器中SP/NSP/SU/NSU所允许的安全/特 权级别访问看门狗。	读写	0x1
6	<b>DMA:</b> 若值为1, DMA可按照本寄存器中SP/NSP/SU/NSU所允许的安全/ 特权级别访问看门狗。	读写	0x0
5	<b>CORE1:</b> 若值为1, WATCHDOG可由核心1访问, 访问权限受本寄存器中S P/NSP/SU/NSU安全/权限级别的约束。	读写	0x1
4	<b>CORE0:</b> 若值为1, WATCHDOG可由核心0访问, 访问权限受本寄存器中S P/NSP/SU/NSU安全/权限级别的约束。	读写	0x1
3	<b>SP:</b> 若值为1, WATCHDOG 可在安全、特权上下文中被访问。	读写	0x1
2	<b>SU:</b> 若值为1且SP亦被设置, WATCHDOG 可在安全、非特权上下文中被 访问。	读写	0x0
1	<b>NSP:</b> 若值为1, WATCHDOG 可在非安全、特权上下文中被访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU:</b> 若值为1且NSP亦被设置, WATCHDOG可在非安全、非特权上下文中被访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：PSM寄存器

偏移量: 0xdc

### 描述

控制调试器、DMA、核心0及核心1是否可访问PSM, 以及其访问的安全/权限级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 967. PSM 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 如果为1, 则PSM可被调试器访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
6	<b>DMA:</b> 如果为1, 则PSM可被DMA访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x0
5	<b>CORE1:</b> 如果为1, 则PSM可被核心1访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
4	<b>CORE0:</b> 如果为1, 则PSM可被核心0访问, 访问权限受本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别限制。	读写	0x1
3	<b>SP:</b> 如果为1, 则可以从安全且特权的上下文访问PSM。	读写	0x1
2	<b>SU:</b> 如果为1且SP同时设置, 则可以从安全且非特权的上下文访问PSM。	读写	0x0
1	<b>NSP:</b> 如果为1, PSM可由非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 如果为1, 且NSP也被设置, PSM可由非安全非特权上下文访问。  仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：XIP\_CTRL寄存器

偏移: 0xe0

### 描述

控制调试器、DMA、核心0和核心1是否能够访问XIP\_CTRL, 以及它们可在何种安全性/特权级别下访问。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 968. XIP\_CTRL  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 如果为1, XIP_CTRL可被调试器访问, 访问权限受本寄存器中的SP/NSP/SU/NSU所允许的安全性/特权级别限制。	读写	0x1
6	<b>DMA:</b> 如果为1, XIP_CTRL可被DMA访问, 访问权限受本寄存器中的SP/NSP/SU/NSU所允许的安全性/特权级别限制。	读写	0x0
5	<b>CORE1:</b> 如果为1, XIP_CTRL可被核心1访问, 访问权限受本寄存器中的SP/NSP/SU/NSU所允许的安全性/特权级别限制。	读写	0x1
4	<b>CORE0:</b> 若为1, 则XIP_CTRL可由核心0访问, 访问权限依本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别。	读写	0x1
3	<b>SP:</b> 若为1, XIP_CTRL可由安全且具特权的上下文访问。	读写	0x1
2	<b>SU:</b> 若为1, 且SP同样设置, XIP_CTRL可由安全且无特权的上下文访问。	读写	0x0
1	<b>NSP:</b> 若为1, XIP_CTRL可由非安全且具特权的上下文访问。	读写	0x0
0	<b>NSU:</b> 若为1, 且NSP同样设置, XIP_CTRL可由非安全且无特权的上下文访问。 ○ 仅当NSP位被设置时, 此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL: XIP\_QMI 寄存器

偏移量: 0xe4

### 描述

控制调试器、DMA、核心0与核心1对XIP\_QMI的访问权限及其安全/特权访问级别。

默认仅允许安全特权处理器或调试访问。

本寄存器仅能由安全、特权处理器或调试器写入, 惟当NSP位被设置时, NSU位可由非安全、特权环境写入。

表 969. XIP\_QMI  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1, XIP_QMI可由调试器访问, 访问权限依本寄存器中SP/NSP/SU/NSU所允许的安全/特权级别。	读写	0x1
6	<b>DMA:</b> 若值为1, 则XIP_QMI可由DMA访问, 访问权限依据本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别。	读写	0x0
5	<b>CORE1:</b> 若值为1, 则XIP_QMI可由核心1访问, 访问权限依据本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别。	读写	0x1
4	<b>CORE0:</b> 若值为1, 则XIP_QMI可由核心0访问, 访问权限依据本寄存器中SP/NSP/SU/NSU所允许的安全/权限级别。	读写	0x1
3	<b>SP:</b> 若值为1, 则XIP_QMI可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 若值为1, 且SP亦被设置, 则XIP_QMI可从安全非特权上下文访问。	读写	0x0
1	<b>NSP:</b> 若值为1, 则XIP_QMI可从非安全特权上下文访问。	读写	0x0

位	描述	类型	复位
0	<b>NSU:</b> 若值为1，且NSP亦被设置，则XIP_QMI可从非安全非特权上下文访问。 仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## ACCESSCTRL：XIP\_AUX 寄存器

偏移: 0xe8

### 描述

控制调试器、DMA、核心0和核心1是否可以访问XIP\_AUX，以及它们可在本寄存器中允许的安全/特权级别下进行访问。

默认仅允许安全特权访问。

本寄存器仅能由安全、特权处理器或调试器写入，惟当NSP位被设置时，NSU位可由非安全、特权环境写入。

表970。XIP\_AUX  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>DBG:</b> 若为1，调试器可按本寄存器中SP/NSP/SU/NSU允许的安全/特权级别访问XIP_AUX。	读写	0x1
6	<b>DMA:</b> 若为1，DMA可按本寄存器中SP/NSP/SU/NSU允许的安全/特权级别访问XIP_AUX。	读写	0x1
5	<b>CORE1:</b> 若为1，核心1可按本寄存器中SP/NSP/SU/NSU允许的安全/特权级别访问XIP_AUX。	读写	0x1
4	<b>CORE0:</b> 若为1，核心0可按本寄存器中SP/NSP/SU/NSU允许的安全/特权级别访问XIP_AUX。	读写	0x1
3	<b>SP:</b> 如果为1，XIP_AUX可从安全特权上下文访问。	读写	0x1
2	<b>SU:</b> 如果为1，且SP同时设置，XIP_AUX可从安全非特权上下文访问。	读写	0x0
1	<b>NSP:</b> 如果为1，XIP_AUX可从非安全特权上下文访问。	读写	0x0
0	<b>NSU:</b> 如果为1，且NSP同时设置，XIP_AUX可从非安全非特权上下文访问。 仅当NSP位被设置时，此位才可由非安全、特权环境写入。	读写	0x0

## 10.7. DMA

RP2350系统DMA是一种外围设备，能够对内存执行任意读写操作。这意味着，类似于处理器，必须谨慎维护不同安全域拥有的内存或外围设备之间的隔离。任何处理器上下文均不得访问属于更高级安全上下文的内存或外围设备。DMA通过确保软件不能利用DMA访问更高级安全上下文（包括处理器通过编程DMA来配置DMA的情况）来维护该不变性。

RP2350 将处理器的安全/特权状态扩展至各个 DMA 通道，且 DMA 通过内置的内存保护单元（MPU）过滤其自身的内存访问，该 MPU 具备类似 Armv8-M SAU 或 RISC-V PMP 的功能。正确配置后，允许多个安全域透明且安全地共享 DMA

资源。若不需此细粒度配置，也可通过 ACCESSCTRL 寄存器（第10.6节）将整个 DMA 模块整体分配给单一安全域。

本节概述了 DMA 的安全特性。具体硬件细节见第[12.6.6](#)节。

### 10.7.1. 通道安全属性

每个通道通过从 SECCFG\_CH0 开始的每通道寄存器分配安全级别。此级别定义：

- 配置和控制通道或监视其状态所需的最低特权级别
- 通道进行内存访问时使用的总线特权级别

为比较安全级别，DMA 对 AHB5 安全/特权属性建立以下全序关系：Secure + Privileged > Secure + Unprivileged > Non-secure + Privileged > Non-secure + Unprivileged。

在任一信道的控制寄存器被写入之前，信道的安全级别可自由更改。此后，其安全级别将被锁定，直至DMA块复位，才可更改。复位时，所有信道的安全级别均设为安全且特权（安全级别 = 3，最大值）。

信道 [SECCFG](#)寄存器的作用详尽列于相关DMA文档第12.6.6.1节。

### 10.7.2. 内存保护单元

RP2350 DMA配备内存保护单元，可配置访问最多八个不同地址范围所需的安全/特权级别，另设默认级别覆盖未匹配任何八个范围的地址。所有DMA读写操作的地址均会与MPU地址映射进行核对。若起始信道的安全级别低于映射中规定的级别，则该访问将被过滤。被过滤的访问不会影响下游总线，并将向违规信道返回总线错误。

DMA内存保护单元由从MPU\_CTRL开始的DMA控制寄存器进行配置。详情请参见第12.6.6.3节。

### 10.7.3. DREQ属性

通道不允许与其安全级别以上的外设的DREQ接口，该安全级别由ACCESSCTRL中的外设访问控制确定。此举旨在避免通过安全外设传输的时序推断任何信息，且由于RP2040 DREQ的清除握手信号可能被恶意利用，导致安全DMA通道使目标FIFO溢出并损坏或丢失数据（有关DREQ握手的详细信息，见第[12.6.4.2](#)节）。

DREQ的安全级别由ACCESSCTRL模块的访问寄存器驱动。ACCESSCTRL首先将 [SP](#)与 [SU](#)逻辑与，再将 [NSP](#)与 [SU](#)逻辑与，随后取4位权限掩码中最低有效位置位的索引。该过程生成一个2位整数，此整数将与DMA通道的安全级别进行比较，以决定其是否可与该DREQ接口。

### 10.7.4. IRQ属性

四条共享DMA中断线（IRQ）中的每一条均具有可配置的安全级别。IRQ的安全级别将与通道的安全级别以及访问DMA中断控制寄存器的总线权限进行比较，以确定：

- 是否允许总线访问读写该IRQ对应的 [INTE](#)/[INTF](#)/[INTS](#)寄存器
- 某特定通道是否将在该IRQ的 [INTS](#)寄存器中可见（进而决定该通道是否会触发

该IRQ的断言)

- 某特定通道是否可以通过该IRQ的INTF/INTS寄存器设置或清除其中断挂起标志

总线访问若欲查看或配置某IRQ，其安全级别必须大于或等于该IRQ的安全级别。

IRQ欲观察通道的中断挂起标志，该IRQ的安全级别必须大于或等于该通道的安全级别。因此，若总线欲观察某通道的中断状态，则总线访问的安全级别必须大于或等于该通道的安全级别。

为了使IRQ能够监控通道的中断挂起标志，其安全级别必须大于或等于该通道的安全级别。

仅有一个INTR寄存器。能通过INTR监控和清除的通道中断，是通过比较通道安全级别与INTR总线访问的安全级别决定的。

## 10.8. OTP

RP2350包含8 kB的OTP存储，组织为 $4096 \times 24$ 位行，并配备硬件ECC保护。这是唯一可变的片上非易失性存储。启动签名密钥和解密密钥存储于OTP中，因此其为安全架构的关键组成部分。本节简要概述了OTP硬件保护功能；第13章详细记录了该硬件。

RP2350 OTP子系统在OTP存储阵列之上增加硬件层，以保护敏感内容：

- OTP以128字节页为粒度进行保护（详见第13.5节）。
  - 每个页面可设为完全可访问、只读或完全不可访问
  - 锁定分别针对安全、非安全及引导加载程序访问单独定义
  - 从 PAGE0\_LOCK0 开始编程 OTP 锁定位将永久生效
  - 从 SW\_LOCK0 开始写入寄存器可将锁定状态提升至更严格的限制，直至下一次 OTP 重置
- 用于访问 SBPI 接口的 OTP 控制寄存器仅硬连线为安全访问模式
  - SBPI 接口用于编程 OTP 并配置电源及模拟硬件
- 保护读取别名在读取过程中提供更高保障，防止对 OTP 电源的故意操控（第 13.1.1 节）
- 硬件在启动时读取 OTP 数组以进行安全硬件配置（第 13.3.4 节）
  - 关键标志（第 13.4 节）启用安全启动、激活故障注入检测器并禁用调试功能
  - OTP 硬件访问密钥（第 13.5.2 节）为 OTP 页面提供额外保护
  - 调试密钥（第 3.5.9.2 节）为有条件限制调试访问的附加机制OTP 中还包含了 RP2350 只读存储器的配置

，特别是其安全启动实现。第13.10节列出了所有预定义的OTP数据位置。启动配置存储于第1页，起始于BOOT\_FLAGS0。启动只读存储器可加载并执行存储于OTP中的代码；详见第5.10.7节的启动只读存储器文档以及从OTPBO

OT\_SRC开始的OTP数据列表。启用安全启动时，从OTP加载的代码须满足所有常规的镜像签名与版本管理要求，因此该代码可作为安全启动链的一部分。chain\_image()只读存储器API允许位于OTP中的引导加载程序回调只读存储器，以验证其所加载的下一个启动阶段。

## 10.9. 故障检测器

毛刺检测器检测系统时钟域中建立时间和保持时间裕度的丢失，该情况可能由对系统时钟或核心电源电压的有意外部干扰引起。检测到丢失时，毛刺检测器将触发系统复位，而非允许软件在可能未定义的状态下继续运行。其响应

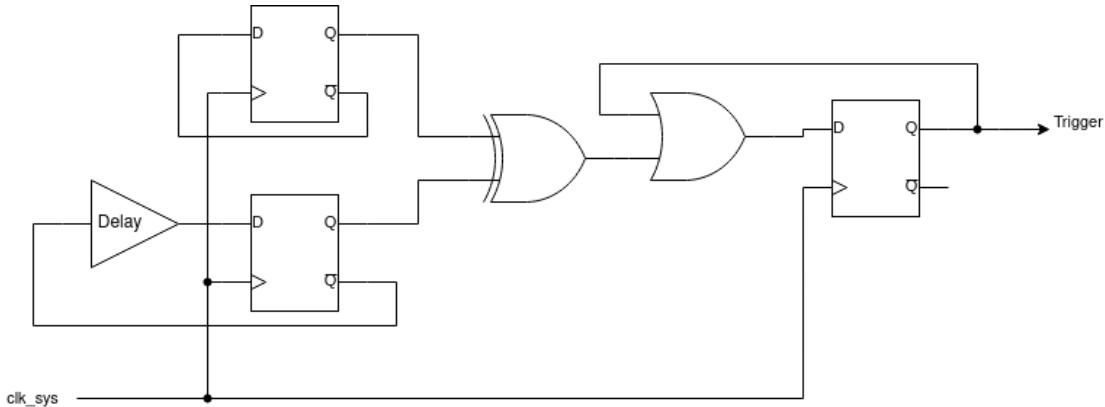
时间在一个系统时钟周期内，区别于欠压检测器，其模拟带宽显著受限。

缺陷检测器默认处于禁用状态，用户可通过在OTP中设置`GLITCH_DETECTOR_ENABLE`标志来激活。出于调试目的，也可通过ARM寄存器启用缺陷检测器。此操作不适用于安全敏感的应用场景，因为系统在软件启用检测器之前存在安全漏洞。

### 10.9.1. 工作原理

缺陷检测器由四个相同的检测电路组成，每个电路基于一对D触发器。这些检测电路分别部署于核心电压域内不同且物理上相距较远的位置。

图43。缺陷检测器触发电路。两组触发器在每个系统时钟周期均切换状态。其中一组在反馈路径中集成了可编程延迟线，而另一组则没有。若设置或保持时间裕度丢失，导致其中一个触发器未能完成切换，两组触发器的输出值将不一致，从而设置触发



检测器在两组D触发器输出值不一致时触发，该情况在正常运行时不应发生。

延迟线的可编程范围为最小系统时钟周期的75%至120%，以15%的增量进行调整。较高的延迟会使电路对设定裕度的损失更为敏感。要配置初始灵敏度，请使用`GLITCH_DETECTOR_SENS` OTP 标志位。您可以通过`SENSITIVITY` 寄存器对每个检测器的灵敏度进行微调。

由于该电路由数字标准单元构成，其传播延迟会紧密追踪电压和温度波动对邻近单元传播延迟所带来的影响。因此，延迟线的传播延迟被指定为最大系统时钟数据路径延迟的一个分数，而非固定纳秒时间。

### 10.9.2. 触发响应

当任一检测器触发时，对应的`TRIG_STATUS` 位将被置位。如果毛刺检测器模块已启动，此检测事件还将复位切换核心域中几乎所有逻辑单元。毛刺检测器启动的条件为：

- DISARM 寄存器未设置为解除启动的位模式，且满足以下至少一项条件：
  - `GLITCH_DETECTOR_ENOTP` 标志在 OTP 模块最近复位前已被编程。
  - ARM 寄存器被设置为启动位模式

这会将大部分切换核心域保持在复位状态约 120 微秒，然后释放复位。具体来说，这会复位 PSM（第 7.3 节），该模块复位所有由 PSM 控制的复位，从处理器冷复位域开始，此外复位所有由 RESETS 模块复位的模块，该模块本身由 PSM 复位。检测器电路以及系统看门狗，包括看门狗擦写寄存器，亦会被复位。

在由毛刺检测器触发的复位后，`CHIP_RESET.HAD_GLITCH_DETECT` 标志会被设置，以便软件能够诊断上次复位是由毛刺检测器触发引起的。检查`TRIG_STATUS` 寄存器，以确定是哪个检测器触发了复位。这对于调整各个检测器的阈值非常有用。

清除检测器电路的唯一方法是复位它们，可通过完全切换核心域复位（例如`RUN` 引脚、SW-DP 复位请求、PoR/BoR 复位，或由`POWMAN` 控制配置的切换核心域复位），或通过启动毛刺检测器模块，使检测器与 PSM 一同复位。

从故障检测器触发中恢复需要低功率振荡器处于运行状态（第8.4节）。允许

当低功率振荡器被禁用时，故障检测器触发会导致芯片持续保持复位状态，直到如 RUN引脚等外部复位信号重置检测器。

### 10.9.3. 寄存器列表

故障检测器控制寄存器的起始地址为 **0x40158000**。

表971。GLITCH\_DETECTOR 寄存器列表

偏移量	名称	说明
0x00	ARM	<p>如果故障检测器尚未被OTP授权，则强制启用故障检测器。启用后，任何单个检测器触发都会导致切换核心电源域的上电复位状态机重新启动。</p> <p>故障检测器触发的事件会累计记录在TRIG_STATUS中。如系统因故障检测器触发而复位，该情况会记录在POWMAN_CHIP_RESET中。</p> <p>该寄存器仅允许安全读写操作。</p>
0x04	解除启用	
0x08	灵敏度	<p>将故障检测器的灵敏度调整为OTP提供默认值以外的其他数值。</p> <p>该寄存器仅允许安全读写操作。</p>
0x0c	LOCK	
0x10	TRIG_STATUS	<p>当检测器输出触发时设置。写入1以清除。</p> <p>(若检测器持续处于失效状态，可能会立即返回高电平。检测器仅能通过完全重置切换的核心电源域予以清除。)</p> <p>该寄存器仅允许安全读写操作。</p>
0x14	TRIG_FORCE	<p>模拟一个或多个检测器的触发。写入1至此寄存器将设置STATUS_TRIG中的相应位。</p> <p>若故障检测器当前处于激活状态，写入1将立即重置切换的核心电源域，并在POWMAN_CHIP_RESET中设置复位原因锁存器，以指示故障检测器复位。</p> <p>该寄存器仅允许安全读写操作。</p>

#### GLITCH\_DETECTOR：ARM寄存器

偏移: 0x00

表972. ARM 寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15:0	<p>如果故障检测器尚未被OTP授权，则强制启用故障检测器。启用后，任何单个检测器触发都会导致切换核心电源域的上电复位状态机重新启动。</p> <p>故障检测器触发的事件会累计记录在TRIG_STATUS中。如系统因故障检测器触发而复位，该情况会记录在POWMAN_CHIP_RESET中。</p> <p>该寄存器仅允许安全读写操作。</p>	读写	0x5bad
	枚举值：		
	0x5bad → 否：不强制激活故障检测器		
	0x0000 → 是：强制激活故障检测器。（除ARM_NO外的任何值均视为是）		

## GLITCH\_DETECTOR：DISARM寄存器

偏移：0x04

表973. DISARM  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p>如故障检测器由OTP激活，则强制解激活。若ARM为是，则忽略此操作。</p> <p>该寄存器仅允许安全读写操作。</p>	读写	0x0000
	枚举值：		
	0x0000 → 否：请勿解除故障检测器的武装。（除DISARM_YES外的任何值均视为否）		
	0xdcaf → 是：解除故障检测器的武装。		

## GLITCH\_DETECTOR：灵敏度寄存器

偏移：0x08

### 说明

将故障检测器的灵敏度调整为OTP提供默认值以外的其他数值。

该寄存器仅允许安全读写操作。

表974。  
灵敏度寄存器

位	描述	类型	复位
31:24	默认	读写	0x00
	枚举值：		
	0x00 → 是：对所有检测器使用OTP中配置的默认灵敏度。（除DEFAULT_NO外的任何值均视为是）		
	0xde → 否：不使用OTP中配置的默认灵敏度，而使用本寄存器中的值。		
23:16	保留。	-	-
15:14	<b>DET3_INV</b> ：必须为DET3的反码，否则将使用默认值。	读写	0x0
13:12	<b>DET2_INV</b> ：必须为DET2的反码，否则将使用默认值。	读写	0x0

位	描述	类型	复位
11:10	<b>DET1_INV</b> : 必须为 DET1 的反码，否则将使用默认值。	读写	0x0
9:8	<b>DET0_INV</b> : 必须为 DET0 的反码，否则将使用默认值。	读写	0x0
7:6	<b>DET3</b> : 设置检测器 3 的灵敏度。较高的数值表示更高的灵敏度。	读写	0x0
5:4	<b>DET2</b> : 设置检测器2的灵敏度。较高的数值表示更高的灵敏度。	读写	0x0
3:2	<b>DET1</b> : 设置检测器1的灵敏度。较高的数值表示更高的灵敏度。	读写	0x0
1:0	<b>DET0</b> : 设置检测器0的灵敏度。较高的数值表示更高的灵敏度。	读写	0x0

## GLITCH\_DETECTOR: LOCK 寄存器

偏移: 0x0c

表 975. *LOCK* 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	写入任意非零值以禁止对ARM、DISARM、SENSITIVITY和LOCK寄存器的写入。该寄存器仅限安全读写。	读写	0x00

## GLITCH\_DETECTOR: TRIG\_STATUS 寄存器

偏移: 0x10

### 描述

当检测器输出触发时设置。写入1以清除。

(若检测器持续处于失效状态，可能会立即返回高电平。检测器仅能通过完全重置切换的核心电源域予以清除。)

该寄存器仅允许安全读写操作。

表 976. *TRIG\_STATUS* 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>DET3</b>	WC	0x0
2	<b>DET2</b>	WC	0x0
1	<b>DET1</b>	WC	0x0
0	<b>DET0</b>	WC	0x0

## GLITCH\_DETECTOR: TRIG\_FORCE 寄存器

偏移: 0x14

表97。  
TRIG\_FORCE 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	<p>模拟一个或多个检测器的触发。写入1至此寄存器将设置STATUS_TRIG中的相应位。</p> <p>若故障检测器当前处于激活状态，写入1将立即重置切换的核心电源域，并在POWMAN_CHIP_RESET中设置复位原因锁存器，以指示故障检测器复位。</p> <p>该寄存器仅允许安全读写操作。</p>	SC	0x0

## 10.10. 工厂测试 JTAG

RP2350包含用于制造后设备测试的JTAG硬件。该接口非公开，但其功能已在此处记录，供用户进行风险评估。

与面向用户的 SWD 调试类似，JTAG 接口在上电时默认禁用，只有在 OTP 上电状态机完成后才被启用。若设置了 CRIT1\_SECURE\_BOOT\_ENABLE、CRIT1\_SECURE\_DEBUG\_DISABLE 或 CRIT1.DEBUG\_DISABLE 标志，JTAG 接口将被无限期保持复位状态，无法通信，也无法控制内部硬件。在设置上述任一关键标志后，唯一重新启用 JTAG 接口的方法是设置 RMA OTP 标志（详见第 10.11 节），但该操作同时永久禁止对用户 OTP 页的读写访问。RMA 标志本身通过第 63 页保护标志进行写保护，可防止不受信任的软件编程该标志。

要将 JTAG 接口从复位状态解除，可通过 SWD 访问并写入 RP-AP 控制寄存器的第 0 位。若要将 JTAG 接口连接至 GPIO (TCK、TMS、TDI、TDO 映射至 GPIO0 → 3)，请设置 RP-AP 控制寄存器的第 1 位。RP-AP 始终可访问，即使外部调试被禁用，因为它也用于输入调试密钥（见3.5.9.2节）。

然而，当任何上述关键 OTP 标志被设置时，尝试移除 JTAG 复位的操作将被忽略。

JTAG 接口提供：

- 标准测试功能，如 IDCODE 和 EXTEST（边界扫描）；这些功能不保证符合 IEEE 标准，因为这是内部工厂测试接口，而非面向用户的调试端口。
- 完整的 AHB 总线访问，具备安全和特权属性，HMASTER 为 3（调试器）。
- 对部分寄存器控制的异步访问，通常限于时钟、振荡器和复位控制。

当 JTAG 接口启用时，其 AHB 总线访问将复用 DMA 读端口。

工厂测试 JTAG 接口的所有细节，除禁用及重新启用该接口的 OTP 标志外，均可随 RP2350 硅片版本的修订而变更。

## 10.11. 退役

返还至 Raspberry Pi Ltd 进行故障分析的设备，必须在返还前完成退役操作，以恢复工厂测试功能。设备退役通过将 OTP PAGE63\_LOCK0.RMA 标志设置为 1 实现。当对大量设备进行系统性问题诊断时，Raspberry Pi Ltd 可能会要求返还设备。

设置 RMA 标志具有以下两项效果：

- 工厂测试的 JTAG 接口将被重新启用，无视任何 CRIT1 标志的状态。
- 第 3 至 61 页将永久无法访问：这些页为未在第 13.10 节中列出预定义内容的全部页。

对 OTP 内容的影响相当于所有内容均提升至不可访问的锁定等级：

- 写入操作将失败；
- 读取操作通过非保护读取别名时返回全 1，通过保护读取别名时产生总线错误。

禁用 OTP 访问的逻辑与重新启用测试接口的逻辑由同一内部信号控制，只要页面 0、1、2、62 或 63 中不存在敏感材料，该位不会为用户 OTP 内容提供外部访问权限。设置 RMA 标志为不可逆操作；如果设备配置为从第 3 至第 61 页存储的 OTP 内容启动，可能导致设备永久无法使用。

设置 RMA 标志后，请测试 OTP 访问（例如通过 SWD 接口），并自行确认存储在 OTP 中的任何敏感数据已被阻止访问。

第 63 页的锁定字仅具备 RMA 功能，因为第 62 和 63 页包含锁定字本身，且每个锁定字均受其自身权限保护。这意味着通过对第 63 页设置硬锁或软锁，可以对 RMA 标志实施写保护。

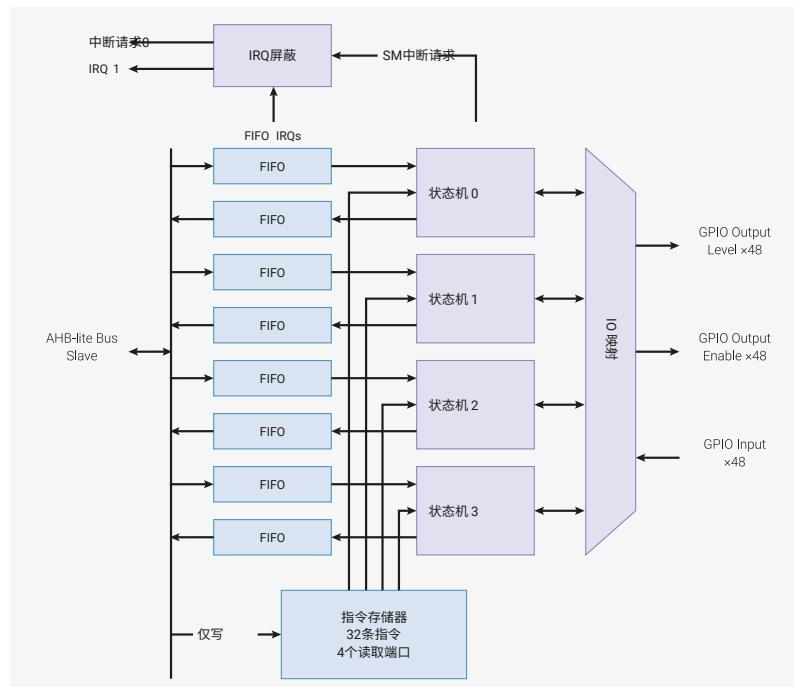
# 第11章 PIO

## 11.1. 概述

RP2350包含3个相同的PIO模块。每个PIO模块均设有专用连接，分别通向总线架构、GPIO及中断控制器。单个PIO模块的示意图如图44所示。

图 44. PIO 模块级示意图。共有三个 PIO 模块，每个模块包含四个状态机。这四个状态机同时从共享指令存储器执行程序。

FIFO 数据队列缓冲 PIO 与系统之间传输的数据。G PIO 映射逻辑允许每个状态机观察并操作最多 32 个 GPIO。



可编程输入/输出模块（PIO）是一种多功能硬件接口。它支持多种 IO 标准，包括：

- 8080 和 6800 并行总线
- I<sub>2</sub>C
- 3 针 I<sub>2</sub>S
- SDIO
- SPI、DSPI、QSPI
- UART
- DPI 或 VGA（通过电阻式 DAC）

PIO 的可编程性与处理器相同。共有三个 PIO 模块，每个含四个状态机。每个状态机均可独立执行顺序程序，以操作 GPIO 并传输数据。与通用处理器不同，PIO 状态机专门用于 IO，强调确定性、精确时序及与固定功能硬件的紧密集成。每个状态机配备：

- 两个 32 位移位寄存器（支持任意方向及任意移位数）
- 两个 32 位暂存寄存器
- 双向各 4×32 位总线 FIFO，可重新配置为单向 8×32 位
- 分数时钟分频器（16 位整数、8 位小数）

- 灵活的 GPIO 映射
- DMA 接口（系统 DMA 可持续吞吐量高达每个时钟周期 1 字）
- IRQ 标志设置／清除／状态

每个状态机及其辅助硬件所占用的硅面积大致相当于一个标准的串行接口模块，例如SPI或I2C控制器。然而，PIO状态机可以动态配置和重新配置，以实现多种不同的接口。

使状态机以类似软件编程的方式进行编程，而非像复杂可编程逻辑器件（CPLD）那样完全可配置的逻辑结构，能够在相同的成本和功耗范围内提供更多的硬件接口。这也为希望直接编程利用PIO全部灵活性的用户提供了更为熟悉的编程模型和更简便的工具流程，而无需依赖PIO库中的预制接口。

PIO不仅灵活，而且性能卓越，这得益于每个状态机内部精心挑选的一组固定功能硬件。当系统时钟为48 MHz时，PIO在输出DPI的活动扫描线期间能够维持360 Mb/s的数据传输速率。在本例中，一个状态机负责帧和扫描线的时序控制，并生成像素时钟。

另一个模块处理像素数据并解码行程长度压缩的扫描线。

状态机的输入与输出最多可映射至32个GPIO（RP2350限制为30个GPIO）。所有状态机均可独立且同时访问任意GPIO。例如，标准UART代码允许 TX、RX、CTS 和 RTS 任意指定为四个GPIO，I2C 对 SDA 和 SCL 亦同。可用自由度取决于特定PIO程序如何使用PIO的引脚映射资源，但至少接口可以在一定GPIO范围内自由上下移动。

### 11.1.1. 与 RP2040 的变化

RP2350新增以下寄存器与控制：

- **DBG\_CFGINFO.VERSION** 表明 PIO 版本，用以运行时检测 PIO 功能。
  - 此 4 位字段在 RP2040 上保留为 0（表示版本 0），在 RP2350 上读取为 1。
- **GPIOBASE** 增加了对每个 PIO 块超过 32 个 GPIO 的支持。
  - 每个 PIO 块仍然限制一次使用 32 个 GPIO，但 GPIOBASE 选择哪 32 个。
- **CTRL.NEXT\_PIO\_MASK** 和 **CTRL.PREV\_PIO\_MASK** 对相邻 PIO 块中的状态机同时应用部分 **CTRL** 寄存器操作。
  - **CTRL.NEXTPREV\_SM\_DISABLE** 同时停止多个 PIO 块中的 PIO 状态机。
  - **CTRL.NEXTPREV\_SM\_ENABLE** 同时启动多个 PIO 块中的 PIO 状态机。
  - **CTRL.NEXTPREV\_CLKDIV\_RESTART** 同步多个 PIO 块中 PIO 状态机的时钟分频器。
- **SM0\_SHIFTCTRL.IN\_COUNT** 将不需要的映射到 IN 的引脚屏蔽为零。
  - 这对于 **MOV x, PINS** 指令非常有用，该指令此前总是返回完整旋转的 32 位值。
- **IRQ0\_INTE** 和 **IRQ1\_INTE** 现在将所有八个 SM 中断标志暴露给系统级中断（不仅限于低四位）。
- 从 RXF0\_PUTGET0 开始的寄存器允许系统对每个 RX FIFO 的内部存储寄存器进行随机读或写访问，
  - 新的 **FJOIN\_RX\_PUTFIFO** 连接模式允许状态机进行随机写入，系统进行随机读取（用于实现状态寄存器）。
  - 新的 **FJOIN\_RX\_GETFIFO** 连接模式允许状态机进行随机读取，系统进行随机写入（用于实现控制寄存器）。
  - 同时设置 **FJOIN\_RX\_PUT** 和 **FJOIN\_RX\_GET** 可使状态机实现随机读写，但系统访问将被禁用。

RP2350 新增了以下指令特性：

- 增加了 `PINCTRL_JMP_PIN` 作为 `WAIT` 指令的来源，偏移范围为 0 至 3。
  - 这使得 `WAIT` 针脚参数实现按状态机独立映射，且不依赖于 IN 映射的针脚。
- 增加了 `PINDIRS` 作为 `MOV` 指令的目的操作数。
  - 这允许通过一条指令更改所有 OUT 映射针脚的方向：`MOV PINDIRS, NULL` 或 `MOV PINDIRS, ~NULL`。
- 增加了状态机中断标志作为 `MOV x, STATUS` 指令的来源。
  - 此功能允许基于 SM IRQ 标志的断言进行分支及阻塞。
- 扩展了 `IRQ` 指令编码，支持状态机设置、清除及监测来自不同 PIO 块的 IRQ 标志。
  - 跨 PIO 的 IRQ 标志无延迟惩罚：一个状态机上的 `IRQ` 可于下一周期被所有状态机观察到。
- 新增了 `FJOIN_RX_GET` FIFO 模式。
  - 一种新的 `MOV` 编码，可将任一四个 RX FIFO 存储寄存器的数据读取至 `OSR`。
  - 该指令允许根据指令位或 Y 临时寄存器索引，对四个 FIFO 条目进行随机读取。
- 新增了 `FJOIN_RX_PUT` FIFO 模式。
  - 一种新的 `MOV` 编码，可将 `ISR` 写入任一四个 RX FIFO 存储寄存器。
  - 寄存器索引由指令位或 Y 临时寄存器指定。

RP2350 新增以下安全功能：

- 非安全 PIO（通过 `ACCESSCTRL` 设置）仅限观察非安全 GPIO。尝试读取安全 GPIO 时将返回 0。
- 禁用非安全 PIO 块（根据 `ACCESSCTRL` 允许非安全访问的）与仅限安全块（不允许访问的）之间的跨 PIO 功能（中断，`CTRL_NEXTPREV` 操作）。

RP2350 包含以下一般性改进：

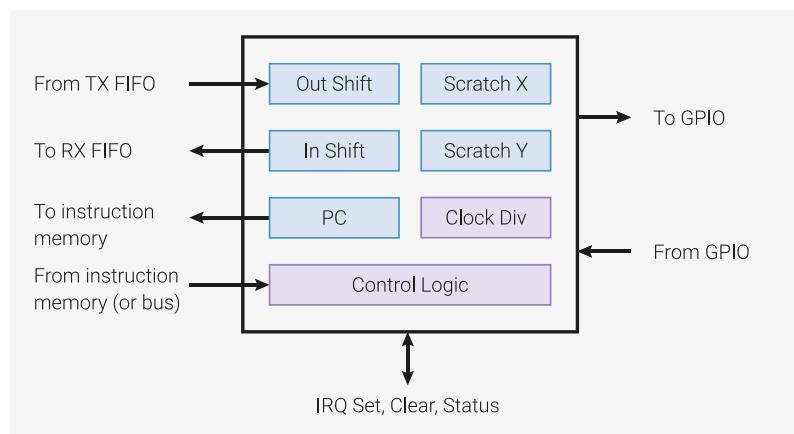
- 将 PIO 块数量从两个增加到三个（8 → 12 个状态机）。
- 改进了 GPIO 输入输出延迟及偏移。
- 相较 RP2040，DMA 请求（DREQ）延迟减少一个周期。

## 11.2. 程序员模型

这四个状态机均从共享指令存储器执行程序。系统软件向该存储器加载程序，配置状态机及 IO 映射，然后启动状态机运行。PIO 程序来源多样：由用户直接汇编、PIO 库提供，或由用户软件程序生成。

从此，状态机通常自治，系统软件通过 DMA、中断和控制寄存器与其交互，类似于 RP2350 上的其他外设。对于更复杂的接口，PIO 提供一组小巧而灵活的原语，允许系统软件更直接控制状态机的控制流。

概述  
数据通过一对FIFO流入和流出。  
状态机执行的程序在这些FIFO、一组内部寄存器与引脚之间传输数据。  
时钟分频器可以将状态机的执行速度按固定倍数降低。



### 11.2.1. PIO 程序

PIO状态机执行简短的二进制程序。

常见接口（如UART、SPI或I2C）的程序可在PIO库中获得。在许多情况下，无需编写PIO程序。然而，直接编程PIO更具灵活性，能够支持设计者可能未预见的多种接口。

PIO共有九条指令：`JMP`、`WAIT`、`IN`、`OUT`、`PUSH`、`PULL`、`MOV`、`IRQ`和`SET`。有关这些指令的详细信息，请参见第11.4节。

虽然PIO总共仅有九条指令，但手动编辑PIO程序二进制文件仍极为困难。PIO汇编是一种文本格式，用于描述PIO程序，其中的每条命令对应于输出二进制文件中的一条指令。以下代码片段展示了一个用PIO汇编语言编写的示例程序：

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.pio> 第8至13行

```

8 .program squarewave
9 set pindirs, 1 ; 将引脚方向设置为输出
10 again:
11 set pins, 1 [1] ; 驱动引脚为高电平，然后延迟一个周期
12 set pins, 0 ; 驱动引脚为低电平
13 jmp again ; 程序计数器跳转至标签`again`
```

PIO汇编器包含于SDK中，名称为 `pioasm`。该程序处理PIO汇编输入文本文件，文件中可包含多个程序，并将组装好的程序写出以供使用。对于SDK，这些组装程序以C头文件的形式输出，包含常量数组。

更多信息，请参见第11.3节。

### 11.2.2. 控制流

在每个系统时钟周期内，每个状态机均取指、译码并执行一条指令。除非明确发生停顿（例如 `WAIT` 指令），否则每条指令恰占用一个周期。指令可在下一条指令执行前插入最长31个周期的延迟，以便编写精确到周期的程序。

程序计数器，简称 `PC`，指向当前周期正在执行的指令存储位置。一般情况下，`PC` 每个周期递增1，并在指令存储末尾回绕。跳转指令为例外，明确指定 `PC` 将取的下一值。

我们的示例汇编程序（上述列出的 `.program squarewave`）演示了这些概念的实际应用。此程序在GPIO引脚上输出占空比为50%、周期为四个时钟周期的方波。利用其他功能（如side-set），周期可缩短至两个时钟周期。

### ⓘ 注意

Side-set 指的是状态机在执行指令的主要副作用之外，驱动少量 GPIO 的操作。详见第11.5.1节。

系统对指令存储器具有写入权限，用以加载程序。时钟分频器以16.8固定点小数形式表示的常数因子，恒定地减缓状态机的执行速度。在下述示例中，若设定了 2.5的时钟分频，方波的周期将为

$4 \times 2.5$  全时钟周期。

此功能对于为串口接口（如UART）设定精确波特率十分有用。

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c> 第34至38行

```

34 // 直接将汇编程序加载至 PIO 指令存储器。
35 // 每个 PIO 实例配备一个32槽的指令存储器，供所有4个状态
36 // 机器可见。系统具有写入权限。
37 for (uint i = 0; i < count_of(squarewave_program_instructions); ++i)
38 pio->instr_mem[i] = squarewave_program_instructions[i];

```

以下代码片段为完整代码示例的一部分，该示例驱动GPIO 0（或我们可能选择映射的其它引脚）输出12.5 MHz方波。我们还可使用引脚 WAIT PIN 指令使状态机执行暂停一段时间，或使用 JMP PIN 指令根据引脚状态进行跳转，从而使控制流依据引脚状态发生变化。

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c> 第42至47行

```

42 // 配置状态机0以sysclk/2.5的频率运行。状态机可
43 // 运作速度最快每个时钟周期可执行一条指令，但我们可以将其
44 // 速度均匀降低，以满足某一精确频率目标，例如
45 // UART 波特率。该寄存器包含 16 位整数除数位和8位
46 // 分数除数位。
47 pio->sm[0].clkdiv = (uint32_t) (2.5f * (1 << 16));

```

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c> 第51至59行

```

51 // 共有五组引脚映射 (out、in、set、side-set、jmp pin)
52 // 不同指令或不同情况下使用这些映射。
53 // 此处仅使用SET指令。配置状态机0的SET指令
54 // 仅作用于GPIO 0；然后配置GPIO0由PIO0控制，
55 // 而不是例如处理器。
56 pio->sm[0].pinctrl =
57 (1 << PIO_SM0_PINCTRL_SET_COUNT_LSB) |
58 (0 << PIO_SM0_PINCTRL_SET_BASE_LSB);
59 gpio_set_function(0, pio_get_funcsel(pio));

```

系统可通过CTRL寄存器随时启动或停止任一状态机。多个状态机可同时启动，且PIO的确定性特性确保其保持完全同步。

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c> 第63至67行

```

63 // 启动状态机。PIO CTRL 寄存器在
64 // 一个PIO实例内为全局寄存器，因此可同时启动/停止多个状态机
65 // 。我们使用该寄存器的硬件原子设置别名来
66 // 在不进行读-改-写操作的情况下，将一个比特位设置为高电平。
67 hw_set_bits(&pio->ctrl, 1 << (PIO_CTRL_SM_ENABLE_LSB + 0));

```

大多数指令从指令存储器执行，但也有其他来源，可以自由混合使用：

- 写入特殊配置寄存器 (**SM<sub>x</sub> INSTR**) 的指令会被立即执行，短暂中断其他指令的执行。例如，向**SM<sub>x</sub> INSTR**写入的**JMP**指令会使状态机从不同位置开始执行。
- 指令可以通过**MOV EXEC**指令从寄存器中执行。
- 指令可以通过**OUT EXEC**指令从输出移位寄存器执行。

最后一种用法极其灵活：指令可以嵌入通过FIFO传输的数据流中。

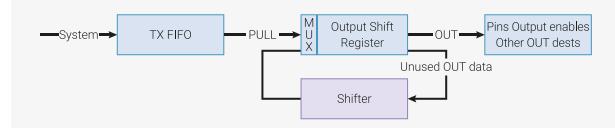
I2C示例利用此功能，嵌入例如 **STOP**和 **RESTART**线路状态，与正常数据并存。在 **MOV**和**OUT EXEC**的情况下，**MOV/OUT**自身在一个周期内执行，而被执行的指令在下一个周期执行。

### 11.2.3. 寄存器

每个状态机拥有少量内部寄存器。这些寄存器保存输入或输出数据以及临时值，例如循环计数变量。

#### 11.2.3.1. 输出移位寄存器 (OSR)

图46。输出移位寄存器 (OSR)。数据一次分配1至32位，未使用的数据通过双向移位器回收利用。一旦清空，OSR将从TX FIFO重新加载。



输出移位寄存器 (OSR) 用于在TX FIFO与引脚或其他目标（如暂存寄存器）之间保存并移位输出数据。

- PULL** 指令：从TX FIFO取出一个32位字，放入OSR。
- OUT** 指令将数据从OSR移位至其他目标，每次移位1至32位。
- 数据移出时，OSR内自动用零填充。
- 启用自动拉取功能时，状态机会在执行**OUT**指令且达到总移位计数阈值后，自动从FIFO重新填充OSR。
- 移位方向可以是左或右，由处理器通过配置寄存器进行设置

例如，要通过FIFO流式传输数据并以每两个时钟周期输出一个字节到引脚：

```

1 .program pull_example1
2 loop:
3 out pins, 8
4 public entry_point:
5 pull
6 out pins, 8 [1]
7 out pins, 8 [1]
8 out pins, 8
9 jmp loop

```

### 11.2.4. 自动拉取

自动拉取（参见第11.5.4节）允许硬件在大多数情况下自动补充OSR，若状态机尝试从空的OSR **OUT**时将暂停执行。此功能具有两个优点：

- 无需使用指令在恰当时机显式从FIFO拉取数据
- 更高吞吐量：只要FIFO保持填充状态，每个时钟周期可输出最多32位

在配置自动拉取后，上述程序可简化为以下形式，且行为完全一致：

```
1 .program pull_example2
2
3 loop:
4 out pins, 8
5 public entry_point:
6 jmp loop
```

程序包裹（第11.5.2节）允许进一步简化，且如有需要，可每个系统时钟周期输出1字节。

```
1 .program pull_example3
2
3 public entry_point:
4 .wrap_target
5 out pins, 8 [1]
6 .wrap
```

#### 11.2.4.1 输入移位寄存器 (ISR)

图47。输入移位寄存器 (ISR)。  
数据每次输入1至32位，当前内容向左或向右移位以腾出空间。  
寄存器内容满后会写入RX FIFO。



- **IN** 指令每次向寄存器移入1至32位数据。
- **PUSH** 指令将ISR内容写入RX FIFO。
- 推送时ISR内容被清零。
- 若启用自动推送，状态机将在达到移位阈值后，于 **IN** 指令时自动推送ISR。
- 移位方向由处理器通过配置寄存器进行配置。

某些外设，如 UART，必须从左侧移位以获得正确的位序，因为线路顺序为最低有效位优先；但处理器可能期望得到的是右对齐的字节。该问题通过特殊的空输入源得到解决，该输入源允许程序员在数据后将一定数量的零移入ISR。

#### 11.2.4.2. 移位计数器

状态机记录通过 **OUT** 指令从OSR移出的位数总和，以及通过 **IN** 指令移入ISR的位数总和。此信息由一对硬件计数器实时跟踪：输出移位计数器和输入移位计数器。每个计数器均可保存0至32（含）之间的值。每次移位操作时，相关计数器按移位数递增，最大值为32（即移位寄存器的宽度）。当计数器达到可配置阈值时，状态机可被配置以执行特定操作：

- 当部分位已被移出后，OSR可自动重新填充（参见第11.5.4节）。
- 当部分位已被移入后，ISR可自动清空（参见第11.5.4节）。

- **PUSH** 或 **PULL** 指令可分别根据输入或输出移位计数器的状态进行条件执行。

在PIO复位或`CTRL_SM_RESTART`信号触发时，输入移位计数器清零（尚未移入任何位），输出移位计数器初始化为32（无剩余位需移出；完全耗尽）。其他若干指令会影响移位计数器：

- 成功执行的 **PULL** 操作将输出移位计数器清零。
- 成功执行的 **PUSH** 操作将输入移位计数器清零。
- `MOV OSR, ...` （即任何写入 `OSR` 的 `MOV` 指令）将输出移位计数器清零。
- `MOV ISR, ...` （即任何写入 `ISR` 的 `MOV` 指令）将输入移位计数器清零。
- `OUT ISR, count` 将输入移位计数器设为 `count`

#### 11.2.4.3. 临时寄存器

每个状态机具有两个32位内部临时寄存器，分别称为 `X` 和 `Y`。

它们的用途如下：

- 作为 `IN/OUT/SET/MOV` 的源或目的地
- 跳转条件的源

例如，假设我们希望为“1”数据位产生长脉冲，为“0”数据位产生短脉冲：

```

1 .program ws2812_led
2
3 public entry_point:
4 pull
5 set x, 23 ; 遍历24位
6 bitloop:
7 set pins, 1 ; 将引脚设为高电平
8 out y, 1 [5] ; 移出1位，并写入y
9 jmp !y skip ; 若该位为0，则跳过额外延迟
10 nop [5]
11 跳过:
12 设置引脚, 0 [5]
13 jmp x-- bitloop ; 如果x非零则跳转，并将x递减
14 jmp entry_point

```

这里 `X` 用作循环计数器，`Y` 用作临时变量，用以基于 `OSR` 中的单个位进行分支。该程序可用于驱动 WS2812 LED 接口，虽然存在更紧凑的实现（最少3条指令）。

`MOV` 允许使用临时寄存器保存/恢复移位寄存器，例如当您希望重复输出相同序列时。

##### **i 注意**

一个更紧凑的WS2812示例（共4条指令）见第11.6.2节。

#### 11.2.4.4. 队列 (FIFOs)

每个状态机配备一对深度为4字的队列，一用于系统向状态机的数据传输（TX），另一用于状态机向系统的数据传输（RX）。TX队列由系统总线主控设备写入，如处理器或DMA控制器，RX队列则由状态机写入。FIFO缓冲区解耦了PIO状态机与系统总线的时序，使状态机能够在更长时间内无需处理器干预而运行。

FIFO缓冲区还会产生数据请求（DREQ）信号，允许系统DMA控制器根据RX FIFO中数据的存在情况或TX FIFO中新数据的可用空间调整其读写速率。这样处理器即可设定一个涉及数千字节数据的长事务，并在无须后续处理器干预的情况下继续执行。

通常，状态机仅在单一方向上传输数据。在此情况下，`SHIFTCTRL_FJOIN`选项可将两个FIFO合并为单向传输的8条目FIFO。这对于如DPI等高带宽接口非常有用。

### 11.2.5. 停滞

状态机可能因多种原因暂时暂停执行：

- 等待（`WAIT`）指令的条件尚未满足
- 当TX FIFO为空时的阻塞型拉取（`PULL`），或当RX FIFO已满时的阻塞型推送（`PUSH`）
- 一条`IRQ_WAIT`指令已设置`IRQ`标志，且正在等待该标志被清除
- 启用自动拉取（`autopull`）且OSR已达到移位阈值时的`OUT`指令
- 启用自动推送（`autopush`），ISR达到移位阈值且RX FIFO已满时的`IN`指令

在此情况下，程序计数器不会前进，状态机将在下一周期继续执行该指令。若指令指定了下一条指令开始前的延迟周期数，则该延迟将在stall解除之后才开始。

#### 注意

附加操作（第11.5.1节）不受停顿影响，且始终在所附指令的第一个周期执行。

### 11.2.6. 引脚映射

PIO控制多达32个GPIO的输出电平和方向，并可监测其输入电平。每个系统时钟周期内，每个状态机可选择不执行、执行一项或执行两项以下操作：

- 通过`OUT`或`SET`指令改变部分GPIO的电平或方向，或通过`IN`指令读取部分GPIO
- 通过副设置操作更改部分GPIO的电平或方向

每个操作使用四个连续GPIO范围之一，每个范围的基址和数量通过各状态机的`PINCTRL`寄存器进行配置。分别为`OUT`、`SET`、`IN`以及副设置操作各设有一个范围。

每个范围可覆盖特定PIO模块可访问的任意GPIO（RP2350上为30个用户GPIO），且这些范围之间可相互重叠。

对于每个单独GPIO输出（电平和方向分别计算），PIO会参考该周期内可能发生的所有8次写操作，并采用编号最高状态机的写入。若同一状态机在同一GPIO上同时执行`SET`/`OUT`与副设置，则以副设置为准。若无任何状态机对该GPIO输出进行写入，其值则沿用上一周期的状态，保持不变。

通常，每个状态机的输出映射至一组独立GPIO，用以实现某种外设接口。

### 11.2.7. IRQ 标志

IRQ标志是状态位，可由状态机或系统设置或清除。共有8个：全部8个对所有状态机可见，且低4位可通过`IRQ0_INTE`和`IRQ1_INTE`控制寄存器屏蔽为PIO的中断请求线路之一。

它们有两个主要用途：

- 由状态机程序发出系统级中断，且可选择等待中断被确认
- 实现两个状态机之间的执行同步

状态机通过 `IRQ` 和 `WAIT` 指令与这些标志交互。

### 11.2.8. 状态机之间的交互

指令存储器实现为1写4读寄存器文件，允许四个状态机在同一周期内同时读取指令而不发生阻塞。  
允许在同一周期内读取指令而无阻塞。

应用多个状态机有三种方式：

- 多个状态机指向相同程序
- 多个状态机指向不同程序
- 使用多个状态机运行同一接口的不同部分，例如UART的发送（TX）和接收（RX）端，或DPI显示的时钟/行同步与像素数据

状态机无法传递数据，但可以通过使用 `IRQ` 标志实现相互同步。总共有8个标志。每个状态机可以使用 `IRQ` 指令设置或清除任意标志，也可以使用 `WAIT IRQ` 指令等待标志变为高电平或低电平。这实现了状态机之间的周期级精确同步。

## 11.3. PIO汇编器（pioasm）

PIO汇编器解析PIO源文件，并生成可供RP2350应用程序包含的汇编版本。此处包括针对SDK构建的C及C++应用程序，以及运行在RP2350 MicroPython移植版上的Python程序。

本节简要介绍可用于 `pioasm` 输入的指令和指令集。有关如何使用 `pioasm`、其在SDK构建系统中的集成、代码透传等扩展功能，以及它支持生成的各种输出格式的详细说明，请参阅Raspberry Pi Pico系列C/C++ SDK。

### 11.3.1. 指令

以下指令用于控制PIO程序的汇编：

`.define (PUBLIC) <symbol> <value>`

定义一个名为 `<symbol>` 的整数符号，值为 `<value>`（参见第11.3.2节）。如果该 `.define` 出现在输入文件中第一个程序之前，则此定义对所有程序全局有效，否则仅对其所在程序局部有效。如指定了 `PUBLIC`，符号将被输出至组装结果，供用户代码使用。对于SDK，其形式如下：

- `#define <program_name> <symbol> value`：用于程序符号
- `#define <symbol> value`：用于全局符号

`.clock_div <divider>`

如果存在该指令，`<divider>` 为该程序状态机的时钟分频器。注意，`divider` 为浮点数值，但当前可能不支持使用算术表达式或定义值。该指令影响程序默认的状态机配置。该指令仅在程序第一条指令之前有效。

`.fifo <fifo_config>`

如果存在该指令，则用于指定程序的FIFO配置。它影响程序的默认状态机配置，但也限制可使用的指令（例如若未配置IN FIFO，则PUSH指令

无意义)。

该指令支持以下配置值：

- **txrx**: TX和RX各有4个FIFO条目；此为默认配置。
- **tx**: 所有8个FIFO条目均分配给TX。
- **rx**: 所有8个FIFO条目均分配给RX。
- **txput**: TX有4个FIFO条目，且有4个FIFO条目用于`mov rxfifo[index], isr`，亦称为 `put`。该值在PIO版本0中不受支持。
- **txget**: TX有4个FIFO条目，且有4个FIFO条目用于`mov osr, rxfifo[index]`，亦称为 `get`。该值在PIO版本0中不受支持。
- **putget**: 4个FIFO条目用于`mov rxfifo[index], isr`亦称为 `put`，另4个FIFO条目用于`mov osr, rxfifo[index]`亦称为 `get`。该值在PIO版本0中不受支持。

该指令仅在程序第一条指令之前有效。

```
.mov_status rxfifo < <n>
.mov_status txfifo < <n>
.mov_status irq <(prev|next)> set <n>
```

该指令配置 `mov` 的 **STATUS** 源。可使用以下三种语法之一设置状态：基于RX FIFO 级别低于值N、TX FIFO 级别低于值N，或在该 PIO 实例（或若指定了 `prev` 或 `next`，则为编号较低或较高的 PIO 实例）上 IRQ 标志 N 被设置。

注意，IRQ 选项要求 PIO 版本为 1。

该指令影响程序默认的状态机配置。该指令仅在程序第一条指令之前有效。

```
.in <count> (left|right) (auto) (<threshold>)
```

若包含此指令，`<count>` 表示使用的 IN 位数。若指定，'left' 或 'right' 控制 ISR 的移位方向；若存在，'auto' 启用“自动推送”；若存在，`<threshold>` 指定“自动推送”阈值。

该指令影响程序的默认状态机配置。

该指令仅在程序中的第一条指令之前有效。当为 PIO 版本 0 汇编时，`<count>` 必须为 32。

```
.program <name>
```

以名称 `<name>` 启动一个新程序。请注意，该名称用于代码中，应为字母数字字符或下划线，且不得以数字开头。程序持续有效，直到遇到另一个 `.program` 指令或源文件结束。PIO 指令仅允许在程序内部使用。

```
.origin <offset>
```

该可选指令用于指定程序必须加载的 PIO 指令内存偏移地址。此指令最常用于必须加载到偏移量 0 的程序，因为它们使用基于数据的 JMP，且（绝对）jmp 目标仅存储在少数几位中。该指令在程序外无效。

```
.out <count> (left|right) (auto) (<threshold>)
```

若存在此指令，`<count>` 表示使用的 OUT 位数。如指定，“left” 或 “right” 控制 OSR 移位方向；如存在，“auto” 启用“自动拉取”；如存在，`<threshold>` 指定“自动拉取”阈值。

该指令影响程序默认的状态机配置。该指令仅在程序第一条指令之前有效。

```
.pio_version <version>
```

该指令设置目标 PIO 硬件版本。RP2350 版本为 1 或 RP2350，且为默认版本号。为兼容 RP2040，可使用 0 或 RP2040。

若该指令出现在输入文件首个程序之前，则该定义为所有程序默认版本，否则指定其所在程序版本。若指定于某程序，须位于该程序首条指令之前。

**.set <count>**

若存在该指令，`<count>`指示 SET 位数。该指令影响程序默认的状态机配置。该指令仅在程序第一条指令之前有效。

**.side\_set <count> (opt) (pindirs)**

若存在该指令，`<count>`指示旁路设置位数。此外，`opt`可被指定以表示指令的某一侧`<value>`为可选（请注意，这需要从指令延迟可用的`<count>`位中额外占用一位）。最后，`pindirs`可被指定以指示侧边设置值应应用于 PINDIRs 而非 PINs。该指令仅在程序中且位于首条指令之前有效。

**.wrap\_target**

该指令置于某条指令之前，指定程序因环绕而继续执行的指令位置。该指令在程序外无效，且程序内仅能使用一次，若未指定，默认指向程序起始位置。

**.wrap**

该指令置于某条指令之后，指定在正常控制流中（即条件跳转`jmp`失败，或无跳转`jmp`情形下）程序环绕至`.wrap_target`指令的位置。该指令在程序外无效，且程序内仅能使用一次，若未指定，默认指向最后一条程序指令之后。

**.lang\_opt <lang> <name> <option>**

指定与特定语言生成器相关的程序选项。（参见Raspberry Pi Pico系列C/C++ SDK中的语言生成器部分）。该指令仅在程序内部有效。

**.word <value>**

将原始16位值作为程序中的指令存储。该指令仅在程序内部有效。

### 11.3.2. 值

以下类型的值可用于定义整数或分支目标：

表978。pioasm中的值，即`<value>`

<b>整数</b>	一个整数值，例如3或-7。
<b>十六进制</b>	一个十六进制值，例如 <code>0xf</code> 。
<b>二进制</b>	一个二进制值，例如 <code>0b1001</code> 。
<b>符号</b>	由 <code>.define</code> 定义的值（参见 <code>pioasm_define</code> ）。
<b>&lt;label&gt;</b>	程序中该标签的指令偏移量。通常与JMP指令配合使用（参见第11.4.2节）。
<b>(&lt;expression&gt;)</b>	一个需求值的表达式；参见表达式部分。请注意括号是必需的。

### 11.3.3. 表达式

表达式可自由用于pioasm值中。

表979。pioasm中的表达式  
i.e. `<expression>`

<code>&lt;expression&gt; + &lt;expression&gt;</code>	两个表达式的和
<code>&lt;expression&gt; - &lt;expression&gt;</code>	两个表达式的差
<code>&lt;expression&gt; * &lt;expression&gt;</code>	两个表达式的乘积
<code>&lt;expression&gt; / &lt;expression&gt;</code>	两个表达式的整数除法
<code>- &lt;expression&gt;</code>	表达式的取负

<expression> << <expression>	一个表达式左移另一个表达式
<expression> >> <expression>	一个表达式右移另一个表达式
:: <expression>	另一个表达式的位反转
<value>	任何值（参见第11.3.2节）

### 11.3.4. 注释

要创建一行注释以忽略某一行中特定符号之后的所有内容，请使用 // 或 ;。

要创建C风格块注释以忽略多行内容，直到出现结束符号，请使用 /\* 开始注释，使用 \*/ 结束注释。

### 11.3.5. 标签

标签在行首使用以下格式：

```
<symbol>:
```

```
PUBLIC <symbol>:
```

#### 提示

标签实际上是一个自动生成的 .define，其值设置为当前程序指令偏移量。PUBLIC 标签以与 PUBLIC .define 相同的方式暴露给用户代码。

### 11.3.6. 指令

所有pioasm指令均遵循以下通用模式：

```
<instruction> (side <side_set_value>) ([<delay_value>])
```

位置：

<instruction> 以下章节详述的汇编指令。（参见第11.4节）

<side\_set\_value> 指令开始时应用于 side\_set 引脚的数值（参见第11.3.2节）。请注意，通过 side <side\_set\_value> 设置的 side-set 值的规则依赖于程序中的 .side\_set 指令（参见 pioasm\_side\_set）。若未指定 side\_set，则 side <side\_set\_value> 无效；若指定了可选数量的 side-set 引脚，则可存在 side <side\_set\_value>；若指定了非可选数量的 side-set 引脚，则必须有 side <side\_set\_value>。<side\_set\_value> 必须在 .side\_set 中指定的 side-set 位数范围内。

指令。

<delay\_value> 指定指令完成后要延迟的周期数。delay\_value 指定为一个数值（参见第11.3.2节），通常取值范围为0至31（含端点）（5位值），但当通过 .side\_set（参见 pioasm\_side\_set）指令启用 sideset 时，位数将减少。如果<delay\_value>未出现，则该指令无延迟。

### ① 注意

pioasm 指令名称、关键字及指令对大小写不敏感；以下汇编语法章节中均使用小写字母，这是 SDK 采用的风格。

### ① 注意

逗号在部分汇编语法章节中出现，但完全可选，例如：out pins, 3 可写作 out pins 3，jmp x-- /label/ 可写成 jmp x--, /label/。以下汇编语法章节均采用每种情况中的第一种风格，因为这符合 SDK 的用法。

## 11.3.7. 伪指令

pioasm 为部分指令提供了别名，以便于使用：

**nop** 汇编为 mov y, y。无副作用，但可作为 side-set 操作或额外延迟的有效手段。

## 11.4. 指令集

### 11.4.1. 概要

PIO 指令长度为 16 位，采用如下编码：

表 980. PIO  
指令编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
跳转	0	0	0	延迟/side-set					条件			地址					
等待	0	0	1	延迟/side-set					极性	源		索引					
输入	0	1	0	延迟/side-set					源			位数					
输出	0	1	1	延迟/side-set					目标			位数					
PUSH	1	0	0	延迟/side-set					0	IfF	块	0	0	0	0		
MOV	1	0	0	延迟/side-set					0	0	0	1	IdxI	0	索引		
PULL	1	0	0	延迟/side-set					1	IfE	块	0	0	0	0		
MOV	1	0	0	延迟/side-set					1	0	0	1	IdxI	0	索引		
MOV	1	0	1	延迟/side-set					目标			Op			源		
IRQ	1	1	0	延迟/side-set					0	Clr	等待	IdxMode		索引			
设置	1	1	1	延迟/side-set					目标			数据					

所有PIO指令均在一个时钟周期内执行。

5位延迟/旁路设置字段的功能取决于状态机的SIDESET\_COUNT配置：

- 最多5位最低有效位（5减SIDESET\_COUNT）编码在本指令与下一指令之间插入的空闲周期数。

- 最多5位最高有效位，由`SIDESET_COUNT`设置，编码旁路设置（第11.5.1节），可在主指令执行的同时将常量置于部分GPIO引脚上。

## 11.4.2. JMP

### 11.4.2.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
跳转	0	0	0	延迟/side-set				条件				地址				

### 11.4.2.2. 操作

如果条件为真，则将程序计数器设置为地址，否则不执行任何操作。

`JMP`指令的延迟周期始终生效，无论条件为真还是假，且在评估条件并更新程序计数器之后发生。

- 条件：

- 000: (无条件)：始终
- 001: `!X`: 寄存器 X 值为零
- 010: `X--`: 寄存器 X 非零，递减前
- 011: `!Y`: 寄存器 Y 值为零
- 100: `Y--`: 寄存器 Y 非零，递减前
- 101: `X!=Y`: 寄存器 X 不等于寄存器 Y
- 110: `PIN`: 基于输入引脚的分支
- 111: `!OSRE`: 输出移位寄存器非空

- 地址：跳转的指令地址。在指令编码中，该地址为 PIO 指令存储器中的绝对地址。

`JMP PIN`根据配置字段`EXECCTRL_JMP_PIN`所选择的 GPIO 引脚进行分支，该字段在最多 32 个可见于状态机的 GPIO 输入中选择一个，独立于状态机的其他输入映射。当该 GPIO 引脚为高电平时，分支跳转将被执行。

`!OSRE` 将自上次执行 `PULL`以来移出的位与由`SHIFTCTRL_PULL_THRESH`配置的移位计数阈值进行比较。这是 autopull（第11.5.4节）使用的相同阈值。

`JMP X--`和`JMP Y--`始终分别递减临时寄存器X或Y。递减操作不依赖于临时寄存器的当前值。分支以寄存器的初始值为条件，即递减发生之前：若寄存器初始值非零，则执行分支。

### 11.4.2.3. 汇编器语法

```
jmp (<cond>) <target>
```

位置：

<cond> 上述可选条件（例如 `!x` 表示临时寄存器X为零）。若未指定条件码，则总执行分支。

<target> 程序标签或数值（参见第11.3.2节），表示程序中指令的偏移量（首条指令偏移量为0）。由于PIO JMP指令在PIO指令存储器中使用绝对地址，JMP需根据程序运行时加载的偏移进行调整。当使用SDK加载程序时，此操作由系统自动处理，但在为`OUT EXEC`编码JMP指令时应格外注意。

### 11.4.3. WAIT

#### 11.4.3.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
等待	0	0	1	延迟侧置				极性	源		索引					

#### 11.4.3.2. 操作

挂起，直到满足指定条件。

与所有挂起指令（第11.2.5节）一样，延迟周期从指令完成之后开始计算。也就是说，若存在任何延迟周期，则延迟计数直到等待条件满足后才开始。

- 极性：

- 1：等待信号为1。
- 0：等待信号为0。

- 源：等待的对象，取值为：

- 00: `GPIO`: 由 `Index` 选择的系统GPIO输入。这是绝对GPIO索引，不受状态机输入IO映射影响。
- 01: `PIN`: 由 `Index` 选择的输入引脚。该状态机先应用输入IO映射，然后由 `Index` 选择映射位中具体等待的位置。换言之，所选引脚为 `PINCTRL_IN_BASE` 配置加上 `Index` 后对32取模所得值。
- 10: `IRQ`: 由 `Index` 选择的PIO IRQ标志
- 11: `JMPPIN`: 等待由 `PINCTRL_JMP_PIN` 配置且索引位于0-3范围内的引脚，均按32取模。`Index` 的其他值保留。

- `Index`: 指定要检查的引脚或位。

`WAIT x IRQ` 的行为与其他 `WAIT` 源略有不同：

- 当 `Polarity` 为1时，状态机在等待条件满足后会清除所选IRQ标志。
- 标志索引的解码方式与 `IRQ` 索引字段相同，从两个最高有效位开始解码（与 `IRQ` 指令中的 `IdxMode` 字段对应）：

- 00: 最低三位用于直接索引该PIO块中的IRQ标志。
- 01 (`PREV`): 该指令引用系统中编号较低的下一个PIO中的IRQ，若当前为PIO0，则回绕至编号最高的PIO。
- 10 (`REL`), 状态机ID (0..3) 通过对两个最低有效位执行模4加法，加入到IRQ索引中。例如，状态机2的标志值为 `0x11` 时，将等待标志3，标志值为 `0x13` 时，将等待标志1。此机制允许多个运行相同程序的状态机相互同步。

- 11 (NEXT)，该指令引用系统中编号紧接其后的PIO的IRQ，若为最高编号的PIO，则回绕至PIO0。

### ⚠ 警告

**WAIT 1 IRQ x**不应与提交给中断控制器的IRQ标志一起使用，以避免与系统中断处理程序发生竞态条件。

#### 11.4.3.3. 汇编语法

```
wait <polarity> gpio <gpio_num>
```

```
wait <polarity> pin <pin_num>
```

```
wait <polarity> irq (prev | next) <irq_num> (rel)
```

```
wait <polarity> jmpin (+ <pin_offset>)
```

位置：

<polarity> 一个数值（参见第11.3.2节），用于指定极性（0或1）。

<pin\_num> 一个数值（参见第11.3.2节），用于指定输入引脚编号（根据SM输入引脚映射）。

<gpio\_num> 一个数值（参见第11.3.2节），用于指定实际GPIO引脚编号。

<irq\_num> (rel) 一个数值（参见第11.3.2节），用于指定待等待的IRQ编号（0-7）。若存在 rel，则实际使用的IRQ编号通过将IRQ编号 ( $irq\_num_{10}$ ) 的低两位替换为IRQ编号与状态机编号之和 ( $irq\_num_{10} + sm\_num_{10}$ ) 的低两位计算得出， $sm\_num_{10}$  为状态机编号。

prev 等待编号较低的下一个PIO模块的IRQ，而非当前PIO模块的IRQ

下一个 等待下一编号的PIO模块上的IRQ，而非当前PIO模块上的IRQ

<pin\_offset> 一个值（参见第11.3.2节），加至 jmp\_pin 以获得实际引脚编号。

#### 11.4.4. IN

##### 11.4.4.1 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
输入	0	1	0	延迟/side-set				源				位数				

#### 11.4.4.2 操作

将Bit count位从 Source移位至输入移位寄存器（ISR）。通过SHIFTCTRL\_IN\_SHIFTDIR为每个状态机配置移位方向。此外，将输入移位计数增加Bit count，最大饱和至32。

- 来源：
  - 000: 引脚
  - 001: X (临时寄存器X)
  - 010: Y (临时寄存器Y)
  - 011: NULL (全零)
  - 100: 保留
  - 101: 保留
  - 110: ISR
  - 111: OSR
- 位计数：要移入ISR的位数。1至32位，其中32编码为 00000

如果启用自动推送，IN在达到推送阈值时也会将ISR内容推入RX FIFO。

(SHIFTCTRL\_PUSH\_THRESH)。IN无论是否发生自动推送，均在一个周期内执行。如果自动推送发生时RX FIFO已满，状态机将暂停。自动推送会将ISR内容清零并重置输入移位计数。详见11.5.4节。

IN始终使用源数据的最低有效位数。例如，若PINCTRL\_IN\_BASE设置为5，指令IN PINS, 3将读取引脚5、6和7的值，并将其移入ISR。首先，ISR向左或向右移位以腾出新输入数据的位置，然后将输入数据复制至该空隙。输入数据的位顺序不依赖于移位方向。

NULL 可用于移位ISR的内容。例如，UART接收最低有效位（LSB）优先，因此必须向右移位。

经过8个输入引脚、1条指令后，输入的串行数据将占据ISR的第31至24位。一条IN NULL, 24条指令会移入24个位的零，使输入数据对齐于ISR的第7至0位。或者，处理器或DMA可以从FIFO地址+3执行字节读取，该操作将获取FIFO内容的第31至24位。

#### 11.4.4.3. 汇编语法

```
in <source>, <bit_count>
```

位置：

<source>

上述指定的源之一。

<bit\_count>

一个值（参见第11.3.2节），指定移位位数（有效范围为1至32）。

#### 11.4.5. OUT

##### 11.4.5.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
输出	0	1	1	延迟/side-set				目标			位数					

### 11.4.5.2. 操作

Shift位计数表示从输出移位寄存器（OSR）移出的位数，并将这些位写入目标。此外，将输出移位计数增加位计数，最大饱和至32。

- 目标：
  - 000: 引脚
  - 001: X (临时寄存器X)
  - 010: Y (临时寄存器Y)
  - 011: NULL (丢弃数据)
  - 100: PINDIRS
  - 101: PC
  - 110: ISR (同时将ISR移位计数器设置为位计数 )
  - 111: EXEC (将OSR移位数据作为指令执行)
- 位计数：表示要从OSR移出的位数，范围为1至32位，32编码为00000

向目标写入一个32位值：低位计数位来自OSR，剩余位填充为零。如果SHIFTCTRL\_OUT\_SHIFTDIR为向右，则该值为OSR的最低有效位计数位，否则为最高有效位。

PINS 和 PINDIRS 使用 OUT 引脚映射，详见第11.5.6节。

若启用自动拉取，当拉取阈值SHIFTCTRL\_PULL\_THRESH达到时，OSR会自动从TX FIFO重新填充。输出移位计数同时被清零为0。在此情况下，如TX FIFO为空，OUT将阻塞，但其他情况下仍在周期内执行。具体内容详见第11.5.4节。

OUT EXEC 允许指令以内联形式包含在FIFO数据流中。OUT本身在一个周期内执行，来自OSR的指令在下一周期执行。该机制执行的指令类型没有限制。初始OUT的延迟周期将被忽略，但执行体可按常规插入延迟周期。

OUT PC 表现为对从OSR移出的地址的无条件跳转。

### 11.4.5.3. 汇编语法

```
out <destination>, <bit_count>
```

位置：

<destination> 上述指定的目标之一。

<bit\_count> 一个值（参见第11.3.2节），指定移位位数（有效范围为1至32）。

## 11.4.6. PUSH

### 11.4.6.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUSH	1	0	0	延迟/side-set				0	IfF	块	0	0	0	0	0	

### 11.4.6.2. 操作

将 ISR 内容作为一个单独的 32 位字推入 RX FIFO，并将 ISR 清零为全零。

- **Iffull**: 如果为 1，除非总输入移位计数达到阈值SHIFTCTRL\_PUSH\_THRESH（与自动推送相同，参见第 11.5.4 节），否则不执行任何操作。
- **Block**: 如果为 1，当 RX FIFO 满时阻止执行。

**PUSH IFFULL**有助于使程序更紧凑，类似自动推送。当启用自动推送会导致 IN在不适当的时间阻塞时，此功能非常有用，例如状态机此时正在断言某些外部控制信号。

PIO 汇编器默认设置 **Block**位。若 **Block**位未设置，**PUSH**指令在RX FIFO已满时不会阻塞，而是立即继续执行下一条指令。此时，FIFO的状态及内容保持不变。ISR依然被清零，且设置了**FDEBUG\_RXSTALL**标志（与阻塞的 **PUSH**或自动推送至满的RX FIFO相同），以指示数据丢失。

#### ① 注意

当SM0\_SHIFTCTRL.FJOIN\_RX\_PUT或FJOIN\_RX\_GET被设置时，**PUSH**指令的操作行为未定义——有关此状态下可使用的 **PUT**和 **GET**指令的详细信息，请参见第11.4.8节和第11.4.9节。

### 11.4.6.3. 汇编语法

```
push (iffull)
```

```
push (iffull) block
```

```
push (iffull) noblock
```

位置：

**iffull** 等同于上述的**IfFull == 1**。即默认情况下，若未指明，则为**IfFull == 0**。

**块** 等同于**Block == 1**如上。如果既未指定 **block**也未指定 **noblock**，则采用此默认设置。

**noblock** 等同于**Block == 0**如上。

### 11.4.7. PULL

#### 11.4.7.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PULL</b>	1	0	0	延迟/side-set				1	IfE	块	0	0	0	0	0	0

### 11.4.7.2. 操作

从 TX FIFO 中加载一个 32 位字到 OSR。

- **IfEmpty**: 若为 1，则除非总输出移位计数达到阈值 **SHIFTCTRL\_PULL\_THRESH**（与 autopull 相同；参见第 11.5.4 节），否则不执行任何操作。
- **Block**: 若为 1，在 TX FIFO 为空时阻塞。若为 0，从空 FIFO 拉取时将暂存 X 复制至 OSR。

某些外设（UART、SPI 等）在无数据可用时应暂停，并在数据到达时继续处理；其他外设（如 I2S）应持续时钟输出，且输出占位数据或重复数据优于停止时钟。这可以通过 **Block** 参数实现。

对空 FIFO 执行非阻塞的 **PULL**，其效果等同于 **MOV OSR, X**。程序可以预先用适当默认值加载临时寄存器 X，或在每次 PULL N **BLOCK** 后执行 **MOV X, OSR**，以便在新数据可用前重复使用最后一个有效的 FIFO 字。

当 TX FIFO 为空且带自动拉取的 OUT 指令会在不当位置停顿时，PULL IFEMPTY 指令非常有用。IfEmpty 允许进行某些与自动拉取相同的程序简化，例如省略外层循环计数器。

但程序中的停顿发生在可控的位置。

#### ① 注意

启用自动拉取时，当 OSR 已满，任何 **PULL** 指令均为无操作，因此 **PULL** 指令表现为屏障。**OUT NULL, 32** 可用于显式丢弃 OSR 内容。详见第 11.5.4.2 节。

### 11.4.7.3. 汇编语法

**pull (ifempty)**

**pull (ifempty) 块**

**pull (ifempty) 非阻塞**

位置：

**ifempty** 等同于 **IfEmpty == 1**。即默认情况下如果未指定，则为 **IfEmpty == 0**。

**块** 等同于 **Block == 1** 如上。如果既未指定 **block** 也未指定 **noblock**，则采用此默认设置。

**noblock** 等同于 **Block == 0** 如上。

### 11.4.8. MOV (至RX)

#### 11.4.8.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MOV</b>	1	0	0	延迟/side-set				0	0	0	1	IdxI	索引			

### 11.4.8.2. 操作

将 ISR 写入所选的 RX FIFO 条目。状态机可以以任意顺序写入 RX FIFO 条目，索引可通过 Y 寄存器或指令中的立即数索引确定。要求设置 SHIFTCTRL\_FJOIN\_RX\_PUT 配置字段，否则其操作未定义。可以通过 .fifo 指令为程序指定 FIFO 配置（参见 pio\_asm\_fifo）。

如果设置了 **IdxI**（立即数索引），RX FIFO 的寄存器由 Index 操作数的最低两位索引。否则，它们由 Y 寄存器的两个最低有效位进行索引。当 IdxI 位清除时，Index 的所有非零值均为保留编码，其操作未定义。

当仅设置 SHIFTCTRL\_FJOIN\_RX\_PUT（位于 SM0\_SHIFTCTRL 至 SM3\_SHIFTCTRL 寄存器中）时，系统还可以通过 RXF0\_PUTGET0 至 RXF0\_PUTGET3 寄存器随机访问 RX FIFO（其中 RXFx 表示访问的是哪个状态机的 FIFO）。在此状态下，FIFO 寄存器存储被重新用作状态寄存器，状态机可随时更新，系统亦可随时读取。例如，正交编码器程序可以始终在状态寄存器中维护当前步进计数，而无需推送至 RX FIFO，从而避免潜在阻塞。

当同时设置 SHIFTCTRL\_FJOIN\_RX\_PUT 与 SHIFTCTRL\_FJOIN\_RX\_GET 时，系统将无法访问 RX FIFO 存储寄存器，但状态机可以任意顺序读写这些寄存器，使其可用作额外的临时存储空间。

#### i 注意

RX FIFO 存储寄存器仅设有单个读端口和写端口，且每个端口在任一时刻仅分配给（系统、状态机）之一。

### 11.4.8.3. 汇编语法

```
mov rxfifo[y], isr
```

```
mov rxfifo[<index>], isr
```

位置：

**y** 字面标记 "y"，表示 RX FIFO 条目通过 Y 寄存器索引。

**<index>** 一个值（参见第 11.3.2 节），指定要写入的 RX FIFO 条目（有效范围为 0-3）。

## 11.4.9. MOV (来自 RX)

### 11.4.9.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOV	1	0	0	延迟/side-set				1	0	0	1	IdxI	索引			

### 11.4.9.2. 操作

将所选 RX FIFO 条目读取至 OSR。PIO 状态机可按任意顺序读取 FIFO 条目，其索引可由 Y 寄存器或指令中的立即数 `Indx` 指定。必须设置`SHIFTCTRL_FJOIN_RX_GET`配置字段，否则其操作未定义。

如果设置了 `Indx`（立即数索引），RX FIFO 的寄存器由 Index 操作数的最低两位索引。否则，它们由Y寄存器的两个最低有效位进行索引。当`Indx`位清除时，Index的所有非零值均为保留编码，其操作未定义。

仅当设置`SHIFTCTRL_FJOIN_RX_GET`时，系统方可通过 RXF0\_PUTGET0 至 RXF0\_PUTGET3（其中 RXFx 表示访问对应状态机的 FIFO）对 RX FIFO 寄存器进行随机访问写入。在此状态下，RX FIFO寄存器存储被重新用作附加配置寄存器，系统可随时更新，状态机亦可随时读取。例如，[UART TX](#)程序可能使用这些寄存器来配置数据位数或额外停止位的存在。

当同时设置`SHIFTCTRL_FJOIN_RX_PUT`与`SHIFTCTRL_FJOIN_RX_GET`时，系统将无法访问RX FIFO存储寄存器，但状态机可以任意顺序读写这些寄存器，使其可用作额外的临时存储空间。

#### ① 注意

RX FIFO 存储寄存器仅设有单个读端口和写端口，且每个端口在任一时刻仅分配给（系统、状态机）之一。

### 11.4.9.3. 汇编语法

```
mov osr, rxfifo[y]
```

```
mov osr, rxfifo[<index>]
```

位置：

`y` 字面标记 "y"，表示 RX FIFO 条目通过 Y 寄存器索引。

`<index>` 一个值（参见第11.3.2节），指定要读取的RX FIFO条目（有效范围0-3）。

## 11.4.10. MOV

### 11.4.10.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<code>MOV</code>	1	0	1	延迟/side-set				目标			Op		源			

### 11.4.10.2. 操作

从源复制数据至目的地。

- 目标：

- 000: PINS (使用与 OUT 相同的引脚映射)
- 001: X (临时寄存器X)
- 010: Y (临时寄存器Y)
- 011: PINDIRS (使用与 OUT 相同的引脚映射)
- 100: EXEC (将数据作为指令执行)
- 101: PC
- 110: ISR (该操作将输入移位计数器复位为0, 即清空)
- 111: OSR (该操作将输出移位计数器复位为0, 即填写满)

• 操作:

- 00: 无
- 01: 取反 (按位取补)
- 10: 位反转
- 11: 保留

• 来源:

- 000: PINS (使用与 IN 相同的引脚映射)
- 001: X
- 010: Y
- 011: NULL
- 100: 保留
- 101: STATUS
- 110: ISR
- 111: OSR

**MOV PC**导致无条件跳转。**MOV EXEC**行为与**OUT EXEC** (第11.4.5节) 相同, 允许将寄存器内容作为指令执行。**MOV**自身执行一个周期, 下一周期执行Source中的指令。**MOV EXEC**的延迟周期被忽略, 但被执行的指令可正常插入延迟周期。

STATUS源的值为全1或全0, 取决于某些状态机状态 (如FIFO满或空), 通过EXECCTRL\_STATUS\_SEL配置。

**MOV**可按有限方式操作所传输的数据, 该操作由 **Operation**参数指定。**Invert**将**Destination**中的每个位设置为对应**Source**位的逻辑非, 即1变0, 0变1。**Bit reverse**将**Destination**中的每个位 n设置为**Source**中第31 - n位, 假设位编号从0至31。

**MOV dst, PINS**通过 **IN**引脚映射读取引脚信号, 掩码位数由**SHIFTCTRL\_IN\_COUNT**指定。读取值的最低有效位对应**PINCTRL\_IN\_BASE**指定的引脚, 后续每个位由编号更高的引脚提供, 超过31后循环。超过**SHIFTCTRL\_IN\_COUNT**配置宽度的结果位将为0。

PIO版本0不支持**MOV PINDIRS, src**指令。

### 11.4.10.3. 汇编语法

```
mov <destination>, (op) <source>
```

位置:

<destination>	上述指定的目标之一。
op	若存在，则为：
	! or ~ 表示 NOT (注：此处始终为按位 NOT)
	:: 表示位反转

## 11.4.11. IRQ

### 11.4.11.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRQ	1	1	0	延迟/side-set				0	Clr	等待	IdxMode	索引				

### 11.4.11.2. 操作

根据参数 Index 设置或清除所选的 IRQ 标志。

- Clear：若为 1，则清除由 Index 选定的标志，而非置位该标志。若设置了 Clear，则 Wait 位无效。
- Wait：若为 1，则挂起直到置位的标志被清除，例如当系统中断处理程序已确认该标志时。
- Index：指定一个 0 到 7 范围内的 IRQ 索引。该 IRQ 标志将根据 Clear 位被置位或清除。
- IdxMode：修改 Index 字段的行为，可修改索引或从不同 PIO 模块索引 IRQ 标志：
  - 00：最低三位用于直接索引该 PIO 块中的 IRQ 标志。
  - 01 (PREV)：指令引用系统中编号更低的 PIO 的 IRQ 标志，若为 PIO0 则循环至编号最高的 PIO。
  - 10 (REL)：状态机 ID (0...3) 通过对两个最低有效位进行模4加法，添加到 IRQ 标志索引中。例如，状态机 2 的标志值为 '0x11' 时，将等待标志 3；标志值为 '0x13' 时，将等待标志 1。此机制允许多个运行相同程序的状态机相互同步。
  - 11 (NEXT)：指令引用系统中编号次高的 PIO 的 IRQ 标志，若该 PIO 为最高编号，则回绕至 PIO0。

所有 IRQ 标志 0-7 均可路由至系统级中断，连接至 PIO 的两个外部中断请求线之一，由 IRQ0\_INTE 和 IRQ1\_INTE 配置。

模加法模式 (REL) 允许对“IRQ”和“WAIT”指令进行相对寻址，以实现运行同一程序的状态机间同步。第 2 位（第三个最低有效位）不受此加法影响。

NEXT/PREV 模式可用于不同 PIO 模块中的状态机之间同步。如果这些状态机的时钟被分频，则其时钟分频器必须相同，且必须通过写入 CTRL\_NEXTPREV\_CLKDIV\_RESTART 以及相关的 NEXT\_PIO\_MASK/PREV\_PIO\_MASK 位实现同步。请注意，根据 ACCESSCTRL，不同对非安全代码可访问的 PIO 之间的跨 PIO 连接将被切断。

如果设置了 Wait，Delay 周期将在等待期结束后才开始。

### 11.4.11.3. 汇编语法

```
irq (prev | next) <irq_num> (rel)
```

```
irq (prev | next) set <irq_num> (rel)
```

```
irq (prev | next) nowait <irq_num> (rel)
```

```
irq (prev | next) wait <irq_num> (rel)
```

```
irq (prev | next) clear <irq_num> (rel)
```

位置：

**<irq\_num> (rel)** 一个值（参见第11.3.2节），指定要定位的IRQ号（0-7）。如果 **rel** 存在，则实际使用的IRQ号通过将IRQ号的最低两位 ( $\text{irq\_num}_{10}$ ) 替换为和的最低两位 ( $\text{irq\_num}_{10} + \text{sm\_num}_{10}$ ) 计算得出，其中  $\text{sm\_n}_{10}$  为状态机编号。

<b>irq</b>	设置IRQ且不等待。
<b>irq 设置</b>	设置IRQ且不等待。
<b>irq 不等待</b>	设置IRQ且不等待。
<b>irq 等待</b>	设置IRQ并等待其被清除后再继续。
<b>irq 清除</b>	清除IRQ。
<b>prev</b>	定位至下一个编号较低的PIO块上的IRQ，而非当前PIO块。
<b>下一个</b>	定位至下一个编号较高的PIO块上的IRQ，而非当前PIO块。

### 11.4.12. SET

#### 11.4.12.1. 编码

位	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>设置</b>	1	1	1	延迟/side-set				目标			数据					

#### 11.4.12.2. 操作

将立即数数据写入目标。

- 目标：

- 000: 引脚
- 001: X (暂存寄存器X) 其最低5位被设置为 Data，其他位全部清零。
- 010: Y (暂存寄存器Y) 其最低5位被设置为 Data，其他位全部清零。
- 011: 保留
- 100: PINDIRS
- 101: 保留
- 110: 保留
- 111: 保留
- Data: 用于驱动引脚或寄存器的5位立即数。

此数值可用于断言控制信号，例如时钟或芯片选择信号，或初始化循环计数器。鉴于 Data 大小为5位，暂存寄存器可以被 SET 为0至31之间的值，足以支持32次迭代的循环。

对 SET 和 OUT 映射至引脚的配置是相互独立的。它们可以映射至不同的引脚位置，例如某引脚用作时钟信号，另一引脚用于数据信号。它们也可以映射至部分重叠的引脚范围：UART发送器可能利用 SET 断言起始位和停止位，同时使用 OUT 指令将FIFO数据发送至同一引脚。

#### 11.4.12.3. 汇编语法

```
set <destination>, <value>
```

位置：

- |               |                              |
|---------------|------------------------------|
| <destination> | 是上述所指定的目标之一。                 |
| <value>       | 设置的值（参见第11.3.2节）（有效范围为0至31）。 |

## 11.5. 功能细节

### 11.5.1. Side-set

侧置功能允许状态机在指令主执行的同时，改变最多5个引脚的电平或方向。

一个必要的例子是高速SPI接口：时钟跳变（切换1→0或0→1）必须与数据跳变同步，新的数据位从OSR移入GPIO。在此情况下，带侧置功能的 OUT 指令能够同时完成这两项操作。

这使接口的时序更加精准，减少整体程序大小（无需单独的 SET 指令切换时钟引脚），并提升SPI的最大运行频率。

侧置功能还使GPIO映射更为灵活，因为其映射独立于 SET 指令。示例 I2C 代码允许将 SDA 和 SCL 映射到任意两个引脚，前提是禁用时钟拉伸功能。通常，SCL 切换以同步数据传输，而 SDA 包含被移出的数据位。然而，某些特定的 I2C 序列，如 Start 和 Stop 线条件，需要在 SDA 和 SCL 上驱动固定模式。I2C 用于实现此目的的映射如下：

- Side-set → SCL

- OUT → SDA
- SET → SDA

这使状态机能够满足两种应用场景：SDA 上传输数据且 SCL 上传输时钟，或在 SDA 和 SCL 上产生固定跳变，同时仍允许将 SDA 和 SCL 映射到任意两个所选 GPIO。

side-set 数据编码于每条指令的Delay/side-set字段内。任意指令均可与 side-set 结合使用，包括写入引脚的指令，如 OUT PINS 或 SET PINS。侧置设置的引脚映射独立于 OUT 和 SET 映射，但可能存在重叠。如果侧置设置与 OUT 或 SET 同时写入同一引脚，将采用侧置设置的数据。

### ① 注意

即使指令停顿，侧置设置仍会立即生效。

```

1 .program spi_tx_fast
2 .side_set 1
3
4 loop:
5 输出引脚, 1位侧置, 0位无
6 jmp loop 侧置 1

```

spi\_tx\_fast示例展示了两个优势：数据与时钟转换可更精确地协同对齐，且程序整体更快速，在本例中每两个系统时钟周期输出一位。程序也可更小。

使用侧置设置时需配置以下四项：

- 用于侧置而非延迟的Delay/side-set字段最高有效位（MSB）的数量。通过设置PINCTRL\_SIDESET\_COUNT进行配置。若设置为5，则无延迟周期可用。如果设置为0，则不会执行side-set。
- 是否使用这些位中的最高有效位作为使能位。仅在使能位为高时执行side-set操作。如果无使能位，且SIDESET\_COUNT非零，则该状态机上的每条指令均执行side-set。此配置由EXECCTRL\_SIDE\_EN参数控制。
- 映射最不重要的side-set位对应的GPIO编号，由PINCTRL\_SIDESET\_BASE配置。
- 确定side-set是写入GPIO电平还是GPIO方向。由EXECCTRL\_SIDE\_PINDIR进行配置。

在上述示例中，我们仅有一个side-set数据位，且每条指令均执行side-set，因此无需使能位。SIDESET\_COUNT为1，SIDE\_EN为false。SIDE\_PINDIR亦为false，因为我们希望驱动时钟的高低电平，而非高低阻抗。SIDESET\_BASE用于选择时钟驱动的GPIO。

## 11.5.2. 程序包装

PIO程序通常包含一个“外循环”：它们重复执行相同的步骤序列，在FIFO与外部世界之间传输数据流。引言中的方波程序是这一概念的最简示例：

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.pio> 第8至13行

```

8 .program squarewave
9 set pindirs, 1 ; 将引脚方向设置为输出
10 again:
11 set pins, 1 [1] ; 驱动引脚为高电平，然后延迟一个周期
12 set pins, 0 ; 驱动引脚为低电平
13 jmp again ; 程序计数器跳转至标签`again`

```

程序主体驱动一个引脚先置为高电平，再置为低电平，产生一个方波周期。整个程序随后循环执行，输出周期性信号。跳转指令本身占用一个周期，每条 `set` 指令也占用一个周期，因此为保持高低电平时长相等，`set pins, 1` 指令增加了一个延迟周期，使状态机在执行 `set pins, 0` 指令前空闲一个周期。总计，每次循环耗时四个周期。这里存在两个缺憾：

- `JMP` 指令占用了指令存储器空间，本应用于其他程序。
- 执行 `JMP` 的额外周期最终导致最大输出速率减半。

由于程序计数器（`PC`）在递增超过31后自然回绕至0，我们可以通过填充整个指令存储器为重复的 `set pins, 1` 和 `set pins, 0` 模式来解决第二个问题，但这样做非常浪费。状态机配备有通过其 `EXECCTRL` 控制寄存器配置的硬件功能，可解决这一常见情况。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave\\_wrap.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave_wrap.pio) 第12至20行

```

12 .program squarewave_wrap
13 ; 类似于squarewave，但使用状态机的.wrap硬件功能替代
14 ; 显式jmp。这是一个免费（0周期）的无条件跳转。
15
16 set pindirs, 1 ; 设置引脚为输出
17 .wrap_target
18 set pins, 1 [1] ; 驱动引脚为高电平，然后延迟一个周期
19 set pins, 0 [1] ; 先驱动引脚为低电平，然后延迟一个周期
20 .wrap

```

执行程序存储器中的指令后，状态机依据以下逻辑更新 `PC`：

1. 若当前指令为 `JMP`，且 `Condition` 为真，则将 `PC` 设置为 `Target`
2. 否则，若 `PC` 等于 `EXECCTRL_WRAP_TOP`，则将 `PC` 设置为 `EXECCTRL_WRAP_BOTTOM`
3. 否则，递增 `PC`；若当前值为31，则将其重置为0。

在 `pioasm` 中，`.wrap_target` 和 `.wrap` 汇编指令实质为标签。该标签导出常量，可分别写入 `WRAP_BOTTOM` 和 `WRAP_TOP` 控制字段：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/generated/squarewave\\_wrap.pio.h](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/generated/squarewave_wrap.pio.h)

```

1 // -----
2 // 此文件由 pioasm 自动生成；请勿修改！ //
3 // -----
4
5 #pragma once
6
7 #include "hardware/pio.h"
8
9 // -----
10 // squarewave_wrap //
11 // -----
12
13 #define squarewave_wrap_target 1
14 #define squarewave_wrap 2
15 #define squarewave_wrap_pio_version 0
16
17 static const uint16_t squarewave_wrap_program_instructions[] = {
18 0xe081, // 0: 设置 pindirs, 1
19 // .wrap_target
20 0xe101, // 1: 设置 pins, 1
21 0xe100, // 2: 设置 pins, 0
22 // .wrap
23 };
24

```

```

25 static const struct pio_program squarewave_wrap_program = {
26 .instructions = squarewave_wrap_program_instructions,
27 .length = 3,
28 .origin = -1,
29 .pio_version = squarewave_wrap_pio_version,
30 .used_gpio_ranges = 0x0
31 #endif
32 };
33
34 static inline pio_sm_config squarewave_wrap_program_get_default_config(uint offset) {
35 pio_sm_config c = pio_get_default_sm_config();
36 sm_config_set_wrap(&c, offset + squarewave_wrap_wrap_target, offset +
37 squarewave_wrap_wrap);
38 return c;
39 }

```

这是来自PIO汇编器 `pioasm` 的原始输出，它已创建一个默认的 `pio_sm_config` 对象，包含程序列表中的 `WRAP` 寄存器值。控制寄存器字段也可以直接初始化。

### 注意

`WRAP_BOTTOM` 和 `WRAP_TOP` 是 PIO 指令存储器中的绝对地址。如果程序加载时存在偏移，包围地址必须相应调整。

`squarewave_wrap` 示例中插入了延迟周期，因此其行为与原始的 `squarewave` 程序完全相同。  
得益于程序包围机制，这些延迟现在可被移除，从而使输出切换速度加倍，同时保持高低电平周期的平衡。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave\\_fast.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave_fast.pio) 第12至18行

```

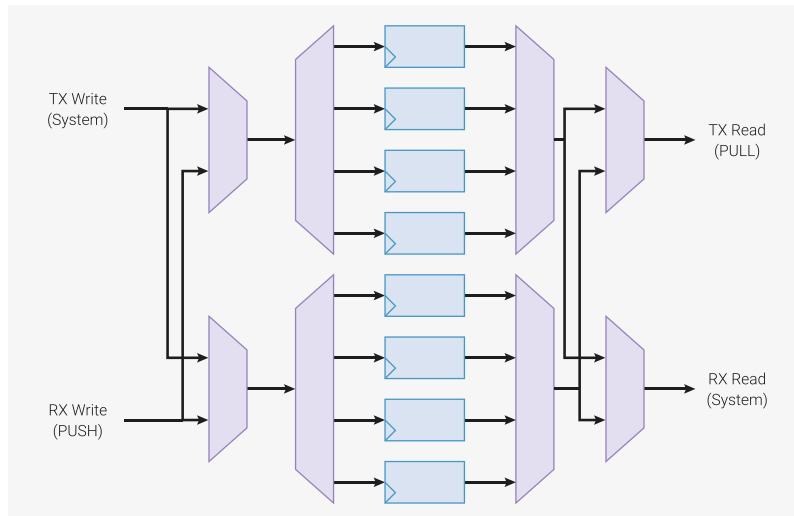
12 .program squarewave_fast
13 ; 类似 squarewave_wrap，但去除了延迟周期，从而使运行速度提升一倍。
14 set pindirs, 1 ; 将引脚设置为输出
15 .wrap_target
16 set pins, 1 ; 使引脚输出高电平
17 set pins, 0 ; 驱动引脚为低电平
18 .wrap

```

### 11.5.3. FIFO 连接

默认情况下，每个状态机在每个方向均配备一个4条目的FIFO：一个用于系统到状态机的数据传输（TX），另一个用于反向传输（RX）。然而，许多应用不需要系统与单个状态机间的双向数据传输，但可能因高带宽接口（如DPI）而受益于更深的FIFO。在这种情况下，`SHIFTCTRL_FJOIN` 可将两个4条目FIFO合并为一个8条目的FIFO。

图48. 可合并的双FIFO。由一对四条目的FIFO组成，采用四个数据寄存器、一个1:4解码器及一个4:1多路复用器实现。额外的复用允许写数据和读数据在TX和RX通道之间交叉，使所有8个条目均可从两个端口访问。



另一个例子是UART：由于UART的TX/CTS和RX/RTS部分是异步的，它们在两台独立的状态机中实现。让每个状态机一半的FIFO资源闲置是一种浪费。将两半合并为仅供TX/CTS状态机使用的TX FIFO，或仅供RX/RTS状态机使用的RX FIFO，使资源得以充分利用。配备8深度FIFO的UART可使中断间隔时间是配备4深度FIFO的UART的两倍。

当一个FIFO的大小增加（从4增至8）时，该状态机上的另一个FIFO容量将减少为零。例如，如果连接到TX，则RX FIFO不可用，任何 **PUSH** 指令将会阻塞。RX FIFO将在 FSTAT 寄存器中同时显示为 RXFULL 和 RXEMPTY。若连接至RX，则情况相反：TX FIFO不可用，且该状态机的 TXFULL 和 TXEMPTY 位均将在 FSTAT 中被置位。同时设置 FJOIN\_RX 和 FJOIN\_TX 将导致两个FIFO均不可用。

8个FIFO条目足以支持以每时钟1字的速度通过RP2350系统DMA前提是DMA未被其他主设备竞争而减慢。

#### **⚠ 警告**

更改 **FJOIN** 将丢弃状态机 FIFO 中存在的任何数据。如该数据不可替代，必须事先清空。

#### 11.5.4. 自动推送与自动拉取

每执行一次 **OUT** 指令，OSR都会逐渐清空，数据被移出。清空后必须重新填充：例如，**PULL** 指令将 TX FIFO 中的一个字移入 OSR。同理，ISR满后也必须清空。一种做法是循环执行，在适当量数据被移出后执行一次 **PULL** 指令：

```

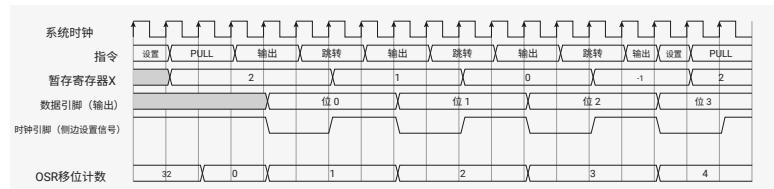
1 .program manual_pull
2 .side_set 1 opt
3
4 .wrap_target
5 set x, 2 ; X = bit count - 2
6 pull side 1 [1] ; 如果无TX数据则在此处暂停
7 bitloop:
8 out pins, 1 side 0 [1] ; 移出数据位并切换时钟信号为低电平
9 jmp x-- bitloop side 1 [1] ; 循环执行3次
10 out pins, 1 side 0 ; 在重新加载X值之前移出最后一位
11 .wrap

```

本程序以每4个周期1位的恒定速率，从每个FIFO字中移出4位数据，并伴随位时钟信号。当TX FIFO为空时，时钟保持高电平暂停（请注意，在指令暂停的周期中侧边设置信号仍然会发生）。

图49展示了状态机执行该程序的过程。

图 49。 manual\_pul  
I 程序的执行。  
。X 用作循环计数器。每次迭代时，移出一位数据，时钟先拉低，然后拉高。每条指令的延迟周期使得每次迭代总共消耗四个周期。  
。第三次循环后，移出第四位数据，状态机立即返回程序起始处，重新加载循环计数器并拉取新数据，同时保持每位四周期的节奏。



该程序存在若干限制：

- 程序占用 5 个指令槽，但只有 2 个指令立即有效（out 引脚指令，1 个置 0 和 1 个置 1），用于输出串行数据和时钟。
- 由于拉取新数据和重新加载循环计数器所需的额外周期，吞吐量限制为系统时钟频率的四分之一。

这是PIO常见的一类问题，因此每个状态机配备了额外硬件以应对。状态机会跟踪OSR的总移位计数 OUT和ISR的 IN，并在这些计数器达到可编程阈值时触发相应操作。

- 在达到或超过拉取阈值的 OUT 指令执行时，若数据可用，状态机可同时从 TX FIFO 重新填充 OSR。
- 在达到或超过推进阈值的 IN 指令执行时，状态机可以将移位结果直接写入 RX FIFO，并清除 ISR。

manual\_pull示例可重写为利用自动拉取（autopull）：

```
1 .program autopull
2 .side_set 1
3
4 .wrap_target
5 out pins, 1 side 0 [1]
6 nop side 1 [1]
7 .wrap
```

该版本比原版更短、更简洁，且若去除延迟周期，运行速度可提高 twice，因为硬件“免费”为OSR完成重新填充。请注意，该程序并不确定下一次拉取前要移位的总位数；硬件在达到可编程阈值 SHIFCTRL\_PULL\_THRESH 时会自动拉取，因此该程序也可从每个FIFO字中移位输出例如16位或32位。

最后，请注意，上述程序与原程序不完全相同，因为它在时钟输出为低电平时暂停，而不是高电平。我们可以通过 PULL I FEMPTY 指令更改暂停位置，该指令使用与自动拉取相同的可配置阈值：

```
1 .program somewhat_manual_pull
2 .side_set 1
3
4 .wrap_target
5 out pins, 1 side 0 [1]
6 pull ifempty side 1 [1]
7 .wrap
```

以下是一个完整示例（PIO程序及用于加载运行该程序的C程序），演示了同一状态机上同时启用自动拉取和自动推送。该程序将状态机0配置为将数据从TX FIFO回环至RX FIFO，吞吐量为每两个时钟周期传输一个字。该示例还展示了当 OSR 和 TX FIFO 均为空时，状态机尝试执行 OUT 操作将会阻塞的情况。

```

1 .program auto_push_pull
2
3 .wrap_target
4 out x, 32
5 in x, 32
6 .wrap

```

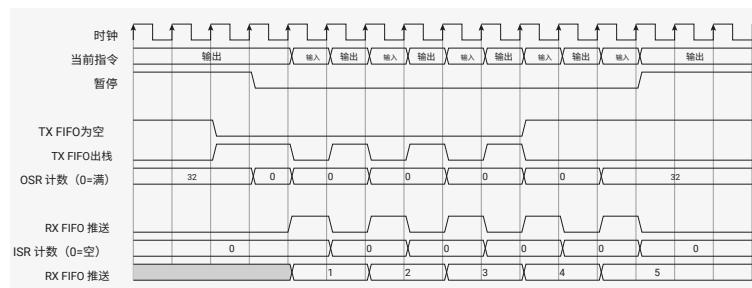
```

1 #include "tb.h" // TODO 该文件基于现有软件树构建, 以支持 printf 等功能
2
3 #include "platform.h"
4 #include "pio_regs.h"
5 #include "system.h"
6 #include "hardware.h"
7
8 #include "auto_push_pull.pio.h"
9
10 int main()
11 {
12 tb_init();
13
14 // 加载程序并配置状态机 0 以实现自动推送 / 拉取,
15 // 阈值为 32, 且程序在边界处环绕。阈值 32
16 // 由寄存器值 00000 编码。
17 for (int i = 0; i < count_of(auto_push_pull_program); ++i)
18 mm_pio->instr_mem[i] = auto_push_pull_program[i];
19 mm_pio->sm[0].shiftctrl =
20 (1u << PIO_SM0_SHIFTCTRL_AUTOPUSH_LSB) |
21 (1u << PIO_SM0_SHIFTCTRL_AUTOPULL_LSB) |
22 (0u << PIO_SM0_SHIFTCTRL_PUSH_THRESH_LSB) |
23 (0u << PIO_SM0_SHIFTCTRL_PULL_THRESH_LSB);
24 mm_pio->sm[0].execctrl =
25 (auto_push_pull_wrap_target << PIO_SM0_EXECCTRL_WRAP_BOTTOM_LSB) |
26 (auto_push_pull_wrap << PIO_SM0_EXECCTRL_WRAP_TOP_LSB);
27
28 // 启动状态机 0
29 hw_set_bits(&mm_pio->ctrl, 1u << (PIO_CTRL_SM_ENABLE_LSB + 0));
30
31 // 将数据推入 TX FIFO, 并从 RX FIFO 弹出
32 for (int i = 0; i < 5; ++i)
33 mm_pio->txf[0] = i;
34 for (int i = 0; i < 5; ++i)
35 printf("%d\n", mm_pio->rx[0]);
36
37 return 0;
38 }

```

图50展示了状态机如何执行示例程序。初始时OSR为空，因此状态机在第一条 OUT指令处暂停。一旦TX FIFO中有数据，状态机会将其传输到OSR。下一周期， OUT指令可以使用OSR中的数据执行（在本例中为将数据传输到X scratch寄存器），状态机同时从FIFO重新填充OSR。由于每条 IN指令会立即填充ISR，ISR始终为空， IN指令直接将数据从scratch X传输到RX FIFO。

图 50。执行 auto\_push\_pull 程序。  
状态机在一个 OUT 指令上停滞，直到数据通过 TX FIFO 进入 OSR。  
随后，OSR 在每次 0 UT 操作时同时被重新填充（由于位数为 32），而 IN 数据绕过 ISR 直接进入 RX FIFO。当 FIFO 被耗尽时，状态机再次停滞，OSR 也再次为空。



为了在正确时机触发自动推送或拉取，状态机使用一对饱和6位计数器来跟踪 ISR 和 OSR 的总移位计数。

- 复位时或在 **CTRL\_SM\_RESTART** 断言时，ISR 移位计数器设为 0（无数据移入），OSR 设为 32（无数据待移出）。
- 一次 **OUT** 指令会将 OSR 移位计数器增加位数。
- 一条 **IN** 指令将 ISR 移位计数器增加位数
- 一条 **PULL** 指令或自动拉取将 OSR 计数器清零
- 一条 **PUSH** 指令或自动推送将 ISR 计数器清零
- 一条 **MOV OSR, n** 或 **MOV ISR, n** 指令分别将 OSR 或 ISR 移位计数器清零
- 一条 **OUT ISR, n** 指令将 ISR 移位计数器设置为 **n**

执行任何 **OUT** 或 **IN** 指令时，状态机将移位计数器与 **SHIFTCTRL\_PULL\_THRESH** 和 **SHIFTCTRL\_PUSH\_THRESH** 值进行比较，以决定是否应采取操作。自动拉取和自动推送由 **SHIFTCTRL\_AUTO\_PULL** 和 **SHIFTCTRL\_AUTO\_PUSH** 字段分别启用。

#### 11.5.4.1. 自动推送详细说明

启用自动推送的 **IN** 指令伪代码如下：

```

1 isr = shift_in(isr, input())
2 isr count = saturate(isr count + in count)
3
4 如果 rx count >= 阈值:
5 如果 rx fifo 已满:
6 暂停
7 else:
8 push(isr)
9 isr = 0
10 isr 计数 = 0

```

硬件在单个机器时钟周期内完成上述步骤，除非发生暂停。

阈值可配置，范围为 1 至 32。

## ❗ 重要

当设置了`SHIFTCTRL_FJOIN_RX_PUT`或`SHIFTCTRL_FJOIN_RX_GET`时，禁止启用自动推送功能。其在该状态下的行为未定义。

### 11.5.4.2. 自动拉取详情

在非`OUT`周期，硬件执行相当于以下伪代码的操作：

```

1 如果是 MOV 或 PULL:
2 osr 计数 = 0
3
4 如果 osr 计数 >= 阈值:
5 如果 tx fifo 非空:
6 osr = pull()
7 osr 计数 = 0

```

因此，自动拉取可在两次 `OUT` 周期之间的任意时刻发生，具体取决于数据何时到达 FIFO。

在 `OUT` 周期，操作序列稍有区别：

```

1 如果 osr 计数 >= 阈值:
2 如果 tx fifo 非空:
3 osr = pull()
4 osr 计数 = 0
5 暂停
6 else:
7 output(osr)
8 osr = shift(osr, out count)
9 osr count = saturate(osr count + out count)
10
11 if osr count >= threshold:
12 如果 tx fifo 非空:
13 osr = pull()
14 osr 计数 = 0

```

硬件能够在移出最后的移位数据的同时重新填充OSR，因为这两个操作可以并行进行。然而，由于这会导致较长的逻辑路径，硬件无法在同一时钟周期内填充空的OSR并 `OUT` 它。

重新填充在某种程度上与您的程序异步，但 `OUT` 操作表现为数据屏障，状态机绝不会 `OUT` 尚未写入 FIFO 的数据。

请注意，当启用自动拉取时，从OSR执行 `MOV` 操作是未定义的；您将根据与系统DMA的竞争情况，读取尚未移出的残余数据或来自FIFO的新字。同样，向OSR执行 `MOV` 操作可能会覆盖刚刚自动拉取的数据。然而，您通过 `MOV` 写入OSR的数据永远不会被覆盖，因为 `MOV` 会更新移位计数器。

如果您确实需要读取OSR内容，应执行某种显式的`PULL`操作。上述所描述的非确定性是硬件自动管理`pull`操作所付出的代价。启用自动`pull`时，`PULL`的行为会改变：若OSR已满，`PULL`将成为无效操作。此举旨在避免与系统DMA的竞争条件。它表现为一个屏障：自动`pull`要么已经发生，此时 `PULL` 无效；要么程序将在`PULL`处阻塞，直到FIFO中出现可用数据。

`PULL`不需类似行为，因为自动`push`不具有相同的非确定性。

### 11.5.5. 时钟分频器

PIO基于系统时钟运行，但该时钟速度对于许多接口而言过快，且可插入的 Delay 周期数有限。某些设备，如UART，要求精确控制和调整信号速率，且理想情况下多个状态机能够在运行相同程序时独立变化。为此，每个状态机都配备了时钟分频器。

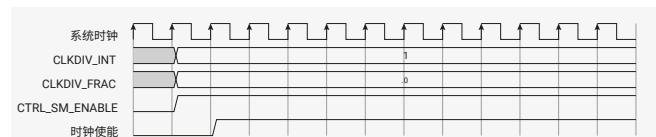
时钟分频器并非降低系统时钟本身的速度，而是重新定义多少个系统时钟周期被视为“一个周期”以便于执行。其通过生成时钟使能信号来实现，该信号可在每个系统时钟周期基础上暂停和恢复执行。时钟分频器以规律的间隔产生时钟使能脉冲，使状态机以某一稳定速度运行，可能远低于系统时钟频率。

通过这种方式实现时钟分频器，可以简化状态机与系统之间的接口，降低延迟，并减小占用面积。当时钟使能信号为低时，状态机完全空闲，但系统仍然可以访问状态机的FIFO并更改其配置。

时钟分频器采用16位整数位和8位小数位，且小数分频器采用一阶delta-sigma调制。时钟除数可在1至65536之间变化，变化步长为  $1/256$ 。

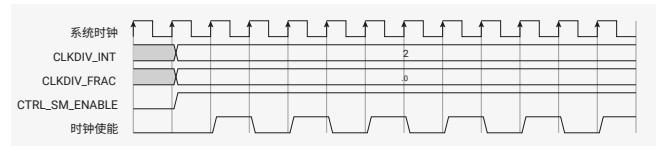
若时钟除数设置为1，状态机将在每个周期运行，即全速运行：

图51。时钟除数为1时状态机的运行情况。一旦通过CTRL寄存器启用状态机，其时钟使能将在每个周期被置位。



通常，整数时钟除数  $n$  将使状态机在每  $n$  个周期运行1次，有效时钟频率为  $f_{sys} / n$ 。

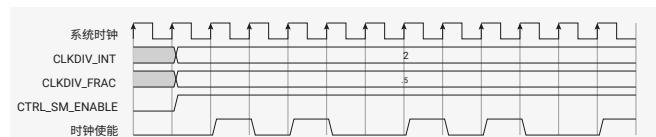
图52。整数时钟除数产生周期性时钟使能信号。时钟分频器不断从  $n$  倒计数，并在



计数至1时发出使能脉冲。小数除法将保持稳定的分频率为  $n + f / 256$ ，其中  $n$  和  $f$  分别是该状态机 CLKDIV 寄存器的整数和小数字段。它通过选择性地将某些分频周期从一个周期延长至下一个周期实现。

$n + 1$

图53。采用平均除数为2.5的分数时钟分频。时钟分频器会累计每个分频周期的分数值，每当该累计值超过1时，下一分频周期的整数除数将增加1。



对于较小的  $n$ ，分数分频器引入的抖动可能无法接受。但对于较大的数值，该效应明显减弱。

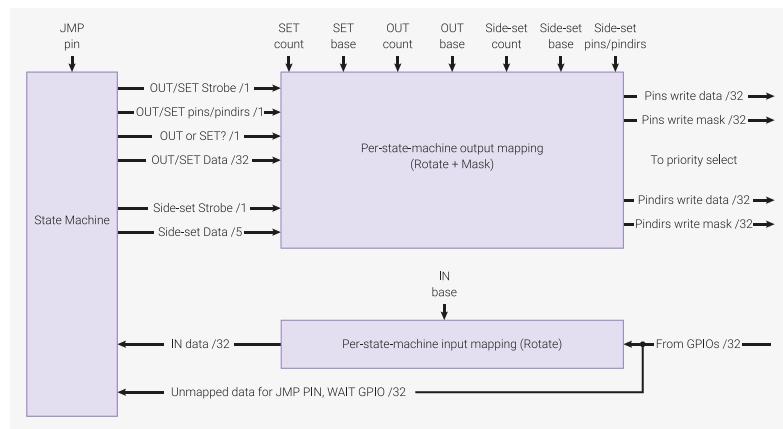
#### 注意

对于高速异步串行通信，建议尽可能使用偶数分频或1 Mbaud的倍数，而非传统的300倍数，以避免不必要的抖动。

### 11.5.6. GPIO 映射

PIO内部设有一个32位寄存器，用于控制其所驱动的每个GPIO的输出电平，另有一个寄存器用于控制输出使能（高电平/高阻抗状态）。在每个系统时钟周期内，每个状态机可向这些寄存器中的部分或全部GPIO输出端口写入数据。

图54。该状态机具有两个独立的输出通道，一个由OUT/SET共享，另一个用于侧置(side-set)（可在任意时刻发生）。三个独立映射（首个GPIO编号及GPIO数量）控制OUT、SET及侧置信号所指向的GPIO。输入数据根据映射至IN数据最低有效位的GPIO进行循环轮换。



输出电平与输出使能寄存器的写入数据及写入掩码来自以下来源：

- 一条 **OUT** 指令可写入最多32位数据。根据指令的 **Destination** 字段，该数据被应用于引脚或引脚方向寄存器 (pindir s)。 **OUT** 数据的最低有效位映射至 **PINCTRL\_OUT\_BASE**，随后映射持续覆盖 **PINCTRL\_OUT\_COUNT** 位，至 **GPIO31** 后循环回绕。
- 一条 **SET** 指令可写入最多5位数据。根据指令的 **Destination** 字段，该数据被应用于引脚或引脚方向寄存器 (pindirs)。 **SET** 数据的最低有效位映射至 **PINCTRL\_SET\_BASE**，随后映射持续覆盖 **PINCTRL\_SET\_COUNT** 位，至 **GPIO31** 后循环回绕。
- 侧置操作可写入最多 5 位。根据寄存器字段 **EXECCTRL\_SIDE\_PINDIR**，此操作应用于引脚或引脚方向。侧置数据的最低有效位映射至 **PINCTRL\_SIDESET\_BASE**，继续映射至 **PINCTRL\_SIDESET\_COUNT** 个引脚；若 **EXECCTRL\_SIDE\_EN** 被设置，则减少一个。

每个 **OUT/SET**/侧置操作写入一段连续的引脚范围，但这些范围在 32 位 GPIO 空间中大小和位置均相互独立。该设计对多种应用均具备足够的灵活性。例如，若一个状态机在一组引脚上实现诸如 SPI 的接口，另一状态机可运行相同程序，映射至不同引脚组，从而提供第二个 SPI 接口。

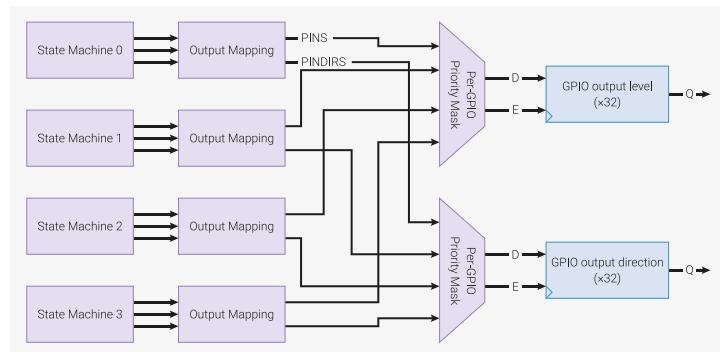
在任一时钟周期中，状态机可执行 **OUT** 或 **SET** 操作，同时进行侧置操作。

引脚映射逻辑为输出电平及输出使能寄存器生成 32 位写掩码与写数据总线，基于该请求及引脚映射配置。

如果侧置组 (side-set) 与同一状态机在同一周期内执行的 **OUT/SET** 操作发生重叠，则侧置组在重叠区域内具有优先权。

### 11.5.6.1. 输出优先级

图55。每个GPIO按状态机优先级选择写掩码。每个GPIO考虑来自四个状态机的电平和方向写入，并应用最高编号状态机的值。



每个状态机可通过其引脚映射硬件，在每个周期内断言一个 **OUT/SET** 和一个侧置组。这为每个状态机生成32位写入数据及写掩码，用于GPIO输出电平和输出使能寄存器。

对于每个GPIO，PIO汇总所有四个状态机的写入操作，并应用最高编号状态机的写入。

状态机。此过程分别针对输出电平和输出值进行——状态机可在同一周期内同时更改同一引脚的电平和方向（例如通过同时执行SET和侧置组），或一个状态机更改GPIO的方向，而另一个状态机更改该GPIO的电平。如果没有状态机声明对GPIO的电平或方向进行写操作，则该值保持不变。

### 11.5.6.2 输入映射

由 **IN** 指令读取的数据映射方式为：最低有效位对应由 **PINCTRL\_IN\_BASE** 选择的 GPIO，随后更高有效位依次对应编号递增的 GPIO，编号超过 31 时循环回绕。

换言之，**IN** 总线是 GPIO 输入值向右旋转 **PINCTRL\_IN\_BASE** 位后的结果。若 GPIO 数量少于 32 个，则 PIO 输入信号以零进行补齐，直至 32 位。

部分指令，如 **WAIT GPIO**，使用绝对 GPIO 编号而非 **IN** 总线索引，此时不进行右旋转操作。

### 11.5.6.3 输入同步器

为防止 PIO 出现亚稳态，每个 GPIO 输入均配备标准的双触发器同步器。这会在输入采样上增加两个周期的延迟，但其优点是状态机可以在任意时刻执行 **IN PINS** 操作，并且只会看到清晰的高电平或低电平，而不会看到可能干扰状态机电路的中间电平。这对于诸如 UART RX 这样的异步接口而言绝对必要。

可以针对每个 GPIO 端口绕过这些同步器。这可以减少输入延迟，但用户必须保证状态机不会在不适当的时间采样其输入。通常这仅适用于诸如 SPI 这类同步接口。通过设置 **INPUT\_SYNC\_BYPASS** 中的相应位可以绕过同步器。

#### ● 警告

采样处于亚稳态的输入可能导致状态机行为不可预测，应当避免。

### 11.5.7. 强制执行及 EXEC 指令

除了指令存储器之外，状态机还可以从另外三种来源执行指令：

- **MOV EXEC** 执行来自某寄存器 **Source** 的指令
- **OUT EXEC** 执行从 OSR 移出的数据
- **SMx\_INSTR** 控制寄存器，系统可向其写入指令以实现即时执行

```

1 .program exec_example
2
3 hang:
4 jmp hang
5 execute:
6 out exec, 32
7 jmp execute
8
9 .program instructions_to_push
10
11 out x, 32
12 in x, 32
13 push

```

```

1 #include "tb.h" // TODO 该文件基于现有软件树构建, 以支持 printf 等功能
2
3 #include "platform.h"
4 #include "pio_regs.h"
5 #include "system.h"
6 #include "hardware.h"
7
8 #include "exec_example.pio.h"
9
10 int main()
11 {
12 tb_init();
13
14 for (int i = 0; i < count_of(exec_example_program); ++i)
15 mm_pio->instr_mem[i] = exec_example_program[i];
16
17 // 启用自动拉取功能, 阈值设置为32
18 mm_pio->sm[0].shiftctrl = (1u << PIO_SM0_SHIFTCTRL_AUTOPULL_LSB);
19
20 // 启动状态机0 —— 程序将停留于 “hang” 循环
21 hw_set_bits(&mm_pio->ctrl, 1u << (PIO_CTRL_SM_ENABLE_LSB + 0));
22
23 // 强制跳转至程序位置 1
24 mm_pio->sm[0].instr = 0x0000 | 0x1; // jmp 执行
25
26 // 向 FIFO 中加载指令和数据混合
27 mm_pio->txf[0] = instructions_to_push_program[0]; // out x, 32
28 mm_pio->txf[0] = 12345678; // 待输出的数据
29 mm_pio->txf[0] = instructions_to_push_program[1]; // in x, 32
30 mm_pio->txf[0] = instructions_to_push_program[2]; // push
31
32 // 推入 TX FIFO 的程序将在 RX FIFO 中返回部分数据
33 while (mm_pio->fstat & (1u << PIO_FSTAT_RXEMPTY_LSB))
34 ;
35
36 printf("%d\n", mm_pio->rxf[0]);
37
38 return 0;
39 }

```

这里将一个示例程序加载到状态机中，该程序执行两个操作：

- 进入无限循环
- 进入一个循环，反复从TX FIFO拉取32位数据，并执行低16位作为指令

C程序启动状态机，状态机随后进入 `hang` 循环。当状态机仍在运行时，C程序强制注入一条 `jmp` 指令，使状态机跳出循环。

当指令写入 `INSTR` 寄存器时，状态机会立即解码并执行该指令，而非从PIO指令存储器取出的指令。程序计数器不前进，因此在下一周期（假设强制写入 `INSTR` 接口的指令未引起阻塞）状态机从中断点继续执行当前程序，除非写入指令本身修改了 `PC` 寄存器。

写入 `INSTR` 寄存器的指令会忽略延迟周期，并立即执行，忽略状态机时钟分频。该接口用于执行初始配置及控制流程变更，因此无论状态机如何配置，均能及时执行指令。

写入 `INSTR` 寄存器的指令允许停滞，状态机将内部锁存该指令直至其执行完成。此状态由 `EXECCTRL_EXEC_STALLED` 标志表示。可通过重启状态

机或向 `INSTR` 写入 `NOP` 指令来清除此状态。

在示例状态机程序的第二阶段，使用了 `OUT EXEC` 指令。`OUT` 指令本身占用一个执行周期，而 `OUT` 执行的指令则在下一个执行周期完成。请注意，我们执行的指令之一亦为 `OUT` —— 状态机在任一周期内仅能执行一个 `OUT` 指令。

`OUT EXEC` 通过将 `OUT` 移位数据写入内部指令锁存器来工作。在下一个周期，状态机会记住必须从此锁存器而非指令存储器执行，并且知道在该第二周期不应推进 `PC`。

该程序运行时将打印 “12345678”。

#### ⚠ 警告

如果写入 `INSTR` 的指令发生阻塞，则会存储在 `OUT EXEC` 和 `MOV EXEC` 共用的指令锁存器中，并覆盖正在执行的指令。如果使用 `EXEC` 指令，则写入 `INSTR` 的指令不得发生阻塞。

## 11.6. 示例

这些示例通过实现常见输入输出接口，展示了 PIO 的一些硬件特性。

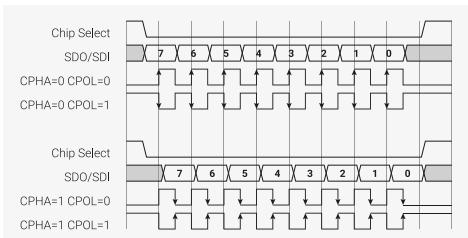
#### 💡 提示

**Raspberry Pi Pico** 系列 **C/C++ SDK** 拥有详尽的 **PIO** 章节，从编写和构建第一个 **PIO** 应用程序开始。后续章节逐行解析部分程序。最后涵盖了更广泛的主题，如结合 **DMA** 使用 **PIO** 及 **PIO** 如何集成到您的软件中。

### 11.6.1. 双工 SPI

图56。在SPI协议中，主机与设备通过一对双向串行数据线交换数据，且同步于时钟（SCK）。两个标志位，CPOL和CPH A，定义了时钟的工作特性。

CPOL表示时钟的空闲状态：0代表低电平，1代表高电平。时钟产生若干脉冲，每个脉冲在两个方向上传输一位数据，但始终返回至其空闲状态。CPH A决定数据采样发生在时钟的哪个边沿：0为前沿，1为后沿。图中箭头指示了主机和设备采样数据时对应的时钟边沿。



SPI是一种常用的串行接口，且拥有复杂的发展历史。以下程序实现了全双工SPI（即数据双向同时传输），CPHA参数设为0。

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio> 第14至32行

```

14 .program spi_cpha0
15 .side_set 1
16
17 ; 引脚分配：
18 ; - SCK 是旁路设置引脚 0
19 ; - MOSI 是输出引脚 0
20 ; - MISO 是输入引脚 0
21 ;
22 ; 必须启用 Autopush 和 Autopull，串行帧大小通过
23 ; 配置推送/拉取阈值来设置。允许左/右移位，但您必须
24 ; 自行对齐数据。对于帧大小为
25 ; 8 位或 16 位，最便捷的方法是利用 RP2040 IO 结构中的
26 ; 窄存储复制和窄加载字节的行为。
27
28 ; 时钟相位=0：数据在每个 SCK 脉冲的上升沿捕获，
29 ; 并在下降沿或首个上升沿之前的某个时间完成转换。
30
31 输出引脚，1 旁路 0 [1] ; 空时暂停（即使旁路设置仍在进行）
32 在引脚中，一侧为1 [1] ; 由于指令阻塞，因此我们在SCK保持低电平时阻塞)

```

该代码使用autopush和autopull持续从FIFO流式传输数据。程序针对转移的每个位执行一次，然后循环。状态机跟踪已移入/移出的位数，并在正确时点自动推动/拉取FIFO。类似程序处理CPHA=1的情况：

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio> 第34至42行

```

34 .program spi_cpha1
35 .side_set 1
36
37 ; 时钟相位=1：数据在每个SCK脉冲的上升沿变化，
38 ; 并在下降沿采样。
39
40 out x, 1 side 0 ; FIFO为空时此处阻塞（保持 SCK未置位）
41 mov pins, x side 1 [1] ; 输出数据，断言SCK（mov pins使用OUT映射）
42 in pins, 1 side 0 ; 输入数据，撤销断言SCK

```

### ⓘ 注意

这些程序不控制片选线；由于不同SPI硬件之间行为差异极大，片选线通常作为软件控制的GPIO实现。上述完整的 `spi.pio` 源代码包含一些示例，演示PIO如何实现硬件片选线。

C语言辅助函数负责配置状态机、连接GPIO并启动状态机运行。请注意，SPI帧大小——即每个FIFO记录传输的比特数——可以在1到32之间任意编程，无需修改程序。配置完成后，状态机即启动运行。

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio> 第46至72行

```

46 static inline void pio_spi_init(PIO pio, uint sm, uint prog_offs, uint n_bits,
47 float clkdiv, bool cpha, bool cpol, uint pin_sck, uint pin_mosi, uint pin_miso) {
48 pio_sm_config c = cpha ? spi_cpha1_program_get_default_config(prog_offs) :
49 spi_cpha0_program_get_default_config(prog_offs);
50 sm_config_set_out_pins(&c, pin_mosi, 1);
51 sm_config_set_in_pins(&c, pin_miso);
52 sm_config_set_sideset_pins(&c, pin_sck);
53 // 本示例代码仅支持 MSB 优先（左移，自动推送/拉取，阈值=位数）
54
55 sm_config_set_out_shift(&c, false, true, n_bits);
56 sm_config_set_in_shift(&c, false, true, n_bits);
57 sm_config_set_clkdiv(&c, clkdiv);
58
59 // MOSI, SCK 输出为低电平, MISO 为输入
60 pio_sm_set_pins_with_mask(pio, sm, 0, (1u << pin_sck) | (1u << pin_mosi));
61 pio_sm_set_pindirs_with_mask(pio, sm, (1u << pin_sck) | (1u << pin_mosi), (1u << pin_sck)
62 | (1u << pin_mosi) | (1u << pin_miso));
63 pio_gpio_init(pio, pin_mosi);
64 pio_gpio_init(pio, pin_miso);
65 pio_gpio_init(pio, pin_sck);
66
67 // 引脚复用器可配置为反转输出（以及其他功能）
68 // 这是一种实现 CPOL=1 的简易方法
69 gpio_set_outover(pin_sck, cpol ? GPIO_OVERRIDE_INVERT : GPIO_OVERRIDE_NORMAL);
70 // SPI 为同步操作，因此绕过输入同步器以减少输入延迟。
71 hw_set_bits(&pio->input_sync_bypass, 1u << pin_miso);
72
73 pio_sm_init(pio, sm, prog_offs, &c);
74 pio_sm_set_enabled(pio, sm, true);
75 }
```

状态机现将立即开始移出出现在 TX FIFO 中的任何数据，并将接收到的数据推送至 RX FIFO。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/pio\\_spi.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/pio_spi.c) 第18至34行

```

18 void __time_critical_func(pio_spi_write8_blocking)(const pio_spi_inst_t *spi, const uint8_t
 *src, size_t len) {
19 size_t tx_remain = len, rx_remain = len;
20 // 对 FIFO 进行 8 位访问，以保证写入数据为字节复制。该
21 // 对于 MSB 优先的移位输出，实现左对齐
22 io_rw_8 *txfifo = (io_rw_8 *) &spi->pio->txf[spi->sm];
23 io_rw_8 *rxfifo = (io_rw_8 *) &spi->pio->rxr[spi->sm];
24 while (tx_remain || rx_remain) {
25 if (tx_remain && !pio_sm_is_tx_fifo_full(spi->pio, spi->sm)) {
26 *txfifo = *src++;
27 --tx_remain;
28 }
29 }
30 }
```

```

29 if (rx_remain && !pio_sm_is_rx_fifo_empty(spi->pio, spi->sm)) {
30 (void) *rxfifo;
31 --rx_remain;
32 }
33 }
34 }
```

综上所述，此完整 C 程序将在 1 MHz 频率下通过 PIO SPI 进行数据回环，支持所有四种 CPOL/CPHA 组合：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi\\_loopback.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi_loopback.c)

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符: BSD-3-Clause
5 */
6
7 #include <stdlib.h>
8 #include <stdio.h>
9
10 #include "pico/stdlib.h"
11 #include "pio_spi.h"
12
13 // 该程序实例化了一个包含所有四种CPOL/CPHA组合的PIO SPI,
14 // 并将串行输入和输出引脚映射到同一GPIO。写入状态机TX FIFO的任何数据
15 // 将被序列化及反序列化，随后重新出现在状态机RX FIFO中。
16 // serialised, deserialised, and reappear in the state machine's RX FIFO.
17
18 #define PIN_SCK 18
19 #define PIN_MOSI 16
20 #define PIN_MISO 16 // 与MOSI相同，实现回环功能
21
22 #define BUF_SIZE 20
23
24 void test(const pio_spi_inst_t *spi) {
25 static uint8_t txbuf[BUF_SIZE];
26 static uint8_t rdbuf[BUF_SIZE];
27 printf("TX:");
28 for (int i = 0; i < BUF_SIZE; ++i) {
29 txbuf[i] = rand() >> 16;
30 rdbuf[i] = 0;
31 printf(" %02x", (int)txbuf[i]);
32 }
33 printf("\n");
34
35 pio_spi_write8_read8_blocking(spi, txbuf, rdbuf, BUF_SIZE);
36
37 printf("RX:");
38 bool mismatch = false;
39 for (int i = 0; i < BUF_SIZE; ++i) {
40 printf(" %02x", (int)rdbuf[i]);
41 mismatch = mismatch || rdbuf[i] != txbuf[i];
42 }
43 if (mismatch)
44 printf("\n Nope\n");
45 else
46 printf("\n OK\n");
47 }
48
49 int main() {
50 stdio_init_all();
```

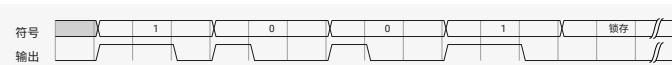
```

51
52 pio_spi_inst_t spi = {
53 .pio = pio0,
54 .sm = 0
55 };
56 float clkdiv = 31.25f; // 1 MHz @ 125 clk_sys
57 uint cpha0_prog_offs = pio_add_program(spi.pio, &spi_cpha0_program);
58 uint cpha1_prog_offs = pio_add_program(spi.pio, &spi_cpha1_program);
59
60 for (int cpha = 0; cpha <= 1; ++cpha) {
61 for (int cpol = 0; cpol <= 1; ++cpol) {
62 printf("CPHA = %d, CPOL = %d\n", cpha, cpol);
63 pio_spi_init(spi.pio, spi.sm,
64 cpha ? cpha1_prog_offs : cpha0_prog_offs,
65 8, // 每个 SPI 帧 8 位
66 clkdiv,
67 cpha,
68 cpol,
69 PIN_SCK,
70 PIN_MOSI,
71 PIN_MISO
72);
73 test(&spi);
74 sleep_ms(10);
75 }
76 }
77 }
```

## 11.6.2. WS2812 LED

WS2812 LED采用专有的脉宽串行格式驱动，宽正脉冲表示“1”位，窄正脉冲表示“0”位。每个LED均具有串行输入和串行输出；LED以链式连接，每个串行输入连接至前一LED的串行输出。

图57. WS2812线  
路格式。宽正  
脉冲表示1，窄正  
脉冲表示0，超长  
负脉冲用于  
锁存使能



LED消耗24位像素数据，随后将任何额外输入数据传递到其输出。通过该方式，单次串行数据传输可独立设置链中每个LED的颜色。长负脉冲将像素数据锁存至LED。

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.pio> 第8至31行

```

8 .program ws2812
9 .side_set 1
10
11 ; 以下常量选取以广泛兼容WS2812、
12 ; WS2812B及SK6812 LED。其他常量可支持
13 ; 特定LED的更高带宽，例如WS2812B LED的(7, 10, 8)。
14
15 .define public T1 3
16 .define public T2 3
17 .define public T3 4
18
19 .lang_opt python sideset_init = pico.PIO.OUT_HIGH
20 .lang_opt python out_init = pico.PIO.OUT_HIGH
21 .lang_opt python out_shiftdir = 1
22
23 .wrap_target
```

```

24 bitloop:
25 out x, 1 side 0 [T3 - 1] ; 即使指令停滞，side-set仍会执行
26 jmp !x do_zero side 1 [T1 - 1] ; 根据移出的位进行分支。正脉冲
27 do_one:
28 jmp bitloop side 1 [T2 - 1] ; 继续将引脚驱动为高电平，产生长脉冲
29 do_zero:
30 nop side 0 [T2 - 1] ; 或将引脚驱动为低电平，产生短脉冲
31 .wrap

```

该程序将只读存储器中的位移入X寄存器，并根据每个位的数据值，在侧置引脚0上产生宽或窄脉冲。必须配置自动拉取，阈值为24。软件随后可将24位像素值写入FIFO，这些值将被串行传输至WS2812 LED链。该 .pio文件包含一个用于配置的C辅助函数：

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.pio> 第36至52行

```

36 static inline void ws2812_program_init(PIO pio, uint sm, uint offset, uint pin, float freq,
37 bool rgbw) {
38 pio_gpio_init(pio, pin);
39 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, true);
40
41 pio_sm_config c = ws2812_program_get_default_config(offset);
42 sm_config_set_sideset_pins(&c, pin);
43 sm_config_set_out_shift(&c, false, true, rgbw ? 32 : 24);
44 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
45
46 int cycles_per_bit = ws2812_T1 + ws2812_T2 + ws2812_T3;
47 float div = clock_get_hz(clk_sys) / (freq * cycles_per_bit);
48 sm_config_set_clkdiv(&c, div);
49
50 pio_sm_init(pio, sm, offset, &c);
51 pio_sm_set_enabled(pio, sm, true);
52 }

```

由于移位是最高有效位优先，且我们的像素大小不是二的幂（因此无法依赖 RP2350 上的窄写复制行为来扩展比特），我们必须对写入 TX FIFO 的值进行预移位处理。

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.c> 第43-45行

```

43 static inline void put_pixel(PIO pio, uint sm, uint32_t pixel_grb) {
44 pio_sm_put_blocking(pio, sm, pixel_grb << 8u);
45 }

```

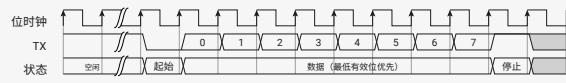
为了通过 DMA 传输像素，我们可以将 autopull 阈值设置为 8 位，将 DMA 传输大小设置为 8 位，并每次向 FIFO 写入一个字节。每个像素将由三个单字节传输组成。由于 RP2350 的总线结构和 DMA 工作方式，DMA 传输的每个字节在写入 32 位 IO 寄存器时会被复制四次，因此数据实际上位于移位寄存器的两端，您可以无忧地向任意方向移位。

### 💡 提示

WS2812 示例是Raspberry Pi Pico 系列 C/C++ SDK 文档中 PIO 章节的教程内容。本教程逐行解析了 ws2812 程序，追踪程序执行流程，并展示了程序各阶段GPIO输出的波形图。

### 11.6.3. UART 发送

图58. UART串行格式。空闲时线路处于高电平。发送器将线路拉低一个位周期以表示串行帧的开始（“起始位”），随后传输一定数量的固定数据位。下一帧串行数据开始前，线路须保持空闲状态至少一个位周期（“停止位”）。



该程序实现了通用异步接收/发送（UART）串行外设的发送功能。更准确地说，可称其为UAT。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_tx/uart\\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio) 第8至18行

```

8 .program uart_tx
9 .side_set 1 opt
10
11 ; 一个8n1 UART发送程序。
12 ; OUT引脚0和side-set引脚0均映射至UART TX引脚。
13
14 下拉 side 1 [7] ; 断言停止位，或在线路空闲状态下阻塞
15 set x, 7 side 0 [7] ; 预载位计数器，断言起始位持续8个时钟周期
16 bitloop: ; 此循环将运行8次（8n1 UART）
17 输出引脚, 1 ; 从OSR移出1位至第一个OUT引脚
18 jmp x-- bitloop [6] ; 每次循环迭代为8个周期。

```

按此编写，程序将执行：

1. 在数据出现前，保持引脚高电平阻塞（注意side-set即使状态机阻塞时亦生效）
2. 断言起始位，持续8个状态机执行周期
3. 移出8位数据，每位持续8个周期
4. 在断言下一个起始位之前，至少保持线路空闲状态8个周期

如果状态机的时钟分频器配置为以期望波特率的8倍运行，则每当数据由软件或系统DMA推入TX FIFO时，该程序将发送格式规范的UART串行帧。为支持不同帧大小（不同数据位数），可以将set x, 7 替换为mov x, y，使y临时寄存器成为每个SM的UART帧大小配置寄存器。

SDK中的 .pio文件亦包含该函数，用于程序加载至PIO指令存储器后配置引脚和状态机：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_tx/uart\\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio) 第24至51行

```

24 static inline void uart_tx_program_init(PIO pio, uint sm, uint offset, uint pin_tx, uint
baud) {
25 // 指示 PIO 初始将所选引脚驱动为输出高电平，然后通过 IO 复用器映射 PIO
26 // 到该引脚。
27 pio_sm_set_pins_with_mask64(pio, sm, 1ull << pin_tx, 1ull << pin_tx);
28 pio_sm_set_pindirs_with_mask64(pio, sm, 1ull << pin_tx, 1ull << pin_tx);
29 pio_gpio_init(pio, pin_tx);
30
31 pio_sm_config c = uart_tx_program_get_default_config(offset);

```

```

32
33 // 输出右移, 无自动拉取
34 sm_config_set_out_shift(&c, true, false, 32);
35
36 // 我们将 OUT 与侧边设置均映射至同一引脚, 因为有时
37 // 需通过 OUT 驱动用户数据至该引脚, 有时
38 // 需驱动恒定值 (起始位/停止位)
39 sm_config_set_out_pins(&c, pin_tx, 1);
40 sm_config_set_sideset_pins(&c, pin_tx);
41
42 // 我们只需要 TX, 因此使用深度为8的FIFO!
43 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
44
45 // 状态机每执行8个周期传输1位。
46 float div = (float)clock_get_hz(clk_sys) / (8 * baud);
47 sm_config_set_clkdiv(&c, div);
48
49 pio_sm_init(pio, sm, offset, &c);
50 pio_sm_set_enabled(pio, sm, true);
51 }

```

该状态机配置为在 `out` 指令中向右移位, 因为UART通常以LSB优先发送数据。配置完成后, 状态机将打印推送至TX FIFO 的所有字符。

Pico示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_tx/uart\\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio) 第53至55行

```

53 static inline void uart_tx_program_putc(PIO pio, uint sm, char c) {
54 pio_sm_put_blocking(pio, sm, (uint32_t)c);
55 }

```

Pico示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_tx/uart\\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio) 第57至60行

```

57 static inline void uart_tx_program_puts(PIO pio, uint sm, const char *s) {
58 while (*s)
59 uart_tx_program_putc(pio, sm, *s++);
60 }

```

SDK中的示例程序将配置一个PIO状态机作为UART TX外设, 并利用其以115200波特率每秒在GPIO 0上打印一条消息。

Pico示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_tx/uart\\_tx.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.c)

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符: BSD-3-Clause
5 */
6
7 #include "pico/stdlib.h"
8 #include "hardware/pio.h"
9 #include "uart_tx.pio.h"
10
11 // 我们将使用PIO在同一GPIO引脚上打印 "Hello, world!"
12 // 通常连接到UART0。
13 #define PIO_TX_PIN 0
14
15 // 检查引脚是否与平台兼容
16 #error 尝试使用平台不支持的 pin >= 32

```

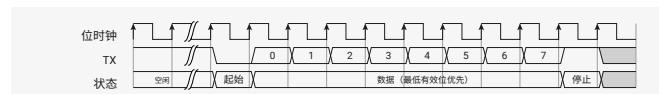
```

17
18 int main() {
19 // 与 Pico 上的默认 UART 波特率相同
20 const uint SERIAL_BAUD = 115200;
21
22 PIO pio;
23 uint sm;
24 uint offset;
25
26 // 此操作将为我们的程序找到一个空闲的 pio 和状态机，并加载它们
27 // 我们使用 pio_claim_free_sm_and_add_program_for_gpio_range (for_gpio_range 变体)
28 // 因此，如果硬件支持，我们将获得适用于地址 gpios >= 32 的 PIO 实例
29
30 bool success = pio_claim_free_sm_and_add_program_for_gpio_range(&uart_tx_program, &pio,
31 &sm, &offset, PIO_TX_PIN, 1, true);
32 hard_assert(success);
33
34 uart_tx_program_init(pio, sm, offset, PIO_TX_PIN, SERIAL_BAUD);
35
36 while (true) {
37 uart_tx_program_puts(pio, sm, "你好，世界！（来自 PIO！）\r\n");
38 sleep_ms(1000);
39 }
40
41 // 这将释放资源并卸载我们的程序
42 pio_remove_program_and_unclaim_sm(&uart_tx_program, pio, sm, offset);
43 }
```

在 RP2350 的两个 PIO 实例基础上，可扩展为 8 个附加 UART 发送接口，分别位于 8 个不同引脚，并使用 8 种不同波特率。

#### 11.6.4. UART 接收

回顾图 58 中显示的 8n1 UART 格式：



我们可以通过等待起始位、以正确时序采样 8 次，并将结果推入 RX FIFO 来恢复数据。以下可能是实现此功能的最简程序：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_rx/uart\\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio) 第 8 至 19 行

```

8 .program uart_rx_mini
9
10 ; 最低可行的8n1 UART接收器。等待起始位，然后采样8位数据
11 ; 以正确的时序。
12 ; 将IN引脚0映射至用作UART RX的GPIO。
13 ; 必须启用自动推送功能，阈值设置为8。
14
15 等待0引脚0 ; 等待起始位
16 将x设为7 [10] ; 预加载位计数器，延迟至第一个数据位的采样窗口
17 bitloop: ; 循环8次
18 从引脚读取, 1 ; 采样数据
19 跳转至x-- bitloop [6] ; 每次迭代耗时8个时钟周期
```

该方法可行，但存在一些问题，例如线路持续低电平时会反复输出 NUL 字符。

理想情况下，我们应舍弃未被起始位和停止位正确帧定的数据（并设置粘性标志以指示此状况），且当线路长时间保持低电平时暂停接收。我们可以将这些指令添加到程序中，代价是增加几条指令。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_rx/uart\\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio) 第44至63行

```

44 .program uart_rx
45
46 ;稍微完善的8n1 UART接收器，可更优雅地处理帧错误和
47 ;中断条件。
48 ;IN引脚0和JMP引脚均映射至作为UART接收的GPIO。
49
50 start:
51 等待引脚0为0 ;阻塞，直到开始位被置位
52 设定x为7 [10]； 预加载位计数器，然后延迟，直到
53 bitloop: ;第一个数据位的中途（包括等待和设定，共12个周期）。
54 从引脚读取1位数据 ;将数据位移入ISR寄存器
55 jmp x-- bitloop [6] ;循环8次，每次循环耗时8个周期
56 jmp 引脚 good_stop ;检查停止位（应为高电平）
57
58 irq 4 相关 ;可能是帧错误或中断信号。设置一个粘滞标志，
59 等待引脚0为低电平 ;并等待线路恢复至空闲状态。
60 jmp start ;如果未检测到正确的帧结构，切勿推送数据。
61
62 good_stop: ;返回 start 之前无延迟；保持一定余量
63 push ;在 TX 时钟稍微偏快的情况下，这一点非常重要。

```

第二个示例未使用自动推送（第11.5.4节），而是采用显式 `push` 指令，以便根据是否检测到正确停止位决定是否推送。该 `.pio` 文件包含一个辅助函数，用于配置状态机并连接带上拉功能的 GPIO 端口：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_rx/uart\\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio) 第67至85行

```

67 static inline void uart_rx_program_init(PIO pio, uint sm, uint offset, uint pin, uint baud) {
68 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, false);
69 pio_gpio_init(pio, pin);
70 gpio_pull_up(pin);
71
72 pio_sm_config c = uart_rx_program_get_default_config(offset);
73 sm_config_set_in_pins(&c, pin); // 用于 WAIT 和 IN
74 sm_config_set_jmp_pin(&c, pin); // 用于 JMP
75 //右移位，禁用自动推送
76 sm_config_set_in_shift(&c, true, false, 32);
77 //由于不进行任何 TX 操作，FIFO 设置更深
78 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_RX);
79 //状态机每 8 次执行周期传输 1 位
80 float div = (float)clock_get_hz(clk_sys) / (8 * baud);
81 sm_config_set_clkdiv(&c, div);
82
83 pio_sm_init(pio, sm, offset, &c);
84 pio_sm_set_enabled(pio, sm, true);
85 }

```

为正确接收按最低有效位优先方式发送的数据，中断服务程序配置为右移位。在左移8位后，遗憾的是，我们的8位数据位位于ISR的31:24位，而最低有效位（LSB）中有24个零。此处的一种选择是执行一条 `in null, 24` 指令，将ISR内容下移至7:0位。另一种方法是从FIFO偏移3字节处进行8位读取，从而使处理器总线硬件（或DMA硬件）自动挑选相关字节，且不额外费力：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_rx/uart\\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio) 第87至93行

```

87 static inline char uart_rx_program_getc(PIO pio, uint sm) {
88 // 从FIFO最高字节进行8位读取，因数据为左对齐
89 io_rw_8 *rxfifo_shift = (io_rw_8*)&pio->rxf[sm] + 3;
90 while (pio_sm_is_rx_fifo_empty(pio, sm))
91 tight_loop_contents();
92 return (char)*rxfifo_shift;
93 }
```

示例程序展示了如何使用该UART RX程序接收由RP2350某硬件UART发送的字符。该程序需将GPIO4连接至GPIO3方能正常工作。为简化对三个不同串口的管理，该程序利用核心1在测试UART（UART 1）上打印字符串，核心0运行的代码则从PIO状态机提取字符，并传递至调试控制台所用的UART（UART 0）。另一种方法是基于中断的IO，采用PIO的FIFO中断请求（IRQ）。若IRQ0\_INTERRUPT寄存器中SM0\_RXNEMPTY位被置位，则每当状态机0的RX FIFO中有字符时，PIO将触发其第一个中断请求信号。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\\_rx/uart\\_rx.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.c)

```

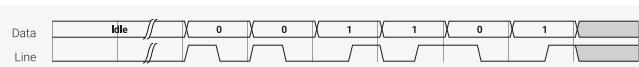
1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符: BSD-3-Clause
5 */
6
7 #include <stdio.h>
8
9 #include "pico/stl.h"
10 #include "pico/multicore.h"
11 #include "hardware/pio.h"
12 #include "hardware/uart.h"
13 #include "uart_rx.pio.h"
14
15 // 本程序
16 // - 使用 UART1 (默认备用 UART) 传输文本
17 // - 通过 PIO 状态机接收该文本
18 // - 将接收的文本打印到默认控制台 (UART0)
19 // 若开发板默认 UART 为 UART1，可能需要进行重新配置。
20 // default UART.
21
22 #define SERIAL_BAUD PICO_DEFAULT_UART_BAUD_RATE
23 #define HARD_UART_INST uart1
24
25 // 需要一根线从 GPIO4 连接至 GPIO3
26 #define HARD_UART_TX_PIN 4
27 #define PIO_RX_PIN 3
28
29 // 检查该引脚是否与平台兼容
30 #error 尝试在不支持该功能的平台上使用编号≥32 的引脚
31
32 // 请求核心1打印字符串，以简化核心0的操作
33 void core1_main() {
34 const char *s = (const char *) multicore_fifo_pop_blocking();
35 uart_puts(HARD_UART_INST, s);
36 }
37
38 int main() {
39 // 控制台输出 (同样是UART，确实可能会让人困惑)
40 setup_default_uart();
41 printf("启动 PIO UART RX 示例\n");
```

```

42 // 配置我们将使用的硬件UART，用于打印字符
43 uart_init(HARD_UART_INST, SERIAL_BAUD);
44 gpio_set_function(HARD_UART_TX_PIN, GPIO_FUNC_UART);
45
46 // 配置我们将用于接收的状态机。
47 PIO pio;
48 uint sm;
49 uint offset;
50
51 // 该操作将为我们的程序找到一个空闲的PIO和状态机并加载它。
52 // 我们使用 pio_claim_free_sm_and_add_program_for_gpio_range (for_gpio_range 变体)
53 // 以便在硬件支持的情况下，获取适用于地址大于等于32的GPIO的PIO实例。
54
55 bool success = pio_claim_free_sm_and_add_program_for_gpio_range(&uart_rx_program, &pio,
56 &sm, &offset, PIO_RX_PIN, 1, true);
57 hard_assert(success);
58
59 uart_rx_program_init(pio, sm, offset, PIO_RX_PIN, SERIAL_BAUD);
60 //uart_rx_mini_program_init(pio, sm, offset, PIO_RX_PIN, SERIAL_BAUD);
61
62 // 指示核心1尽最大速度向uart1打印文本
63 multicore_launch_core1(core1_main);
64 const char *text = "Hello, world from PIO! (Plus 2 UARTs and 2 cores, for complex
65 reasons)\n";
66 multicore_fifo_push_blocking((uint32_t) text);
67
68 // 将从PIO接收到的字符回显至控制台
69 while (true) {
70 char c = uart_rx_program_getc(pio, sm);
71 putchar(c);
72 }
73
74 // 此操作将释放资源并卸载程序
75 pio_remove_program_and_unclaim_sm(&uart_rx_program, pio, sm, offset);
76 }
```

## 11.6.5. 曼彻斯特码串行发送与接收

图59。曼彻斯特串行线路编码。每个数据位由高脉冲后跟低脉冲（表示“0”位）或低脉冲后跟高脉冲（表示“1”位）表示。



Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\\_encoding/manchester\\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio) 第8行至30行

```

8 .program manchester_tx
9 .side_set 1 opt
10
11 ; 每12个周期传输一位。‘0’ 编码为高—低序列
12 ; （每部分持续半个比特周期，即6个周期），‘1’ 编码为
13 ; 低—高序列。
14 ;
15 ; side-set位0必须映射至用于TX的GPIO。
16 ; 必须启用Autopull——本程序不关心阈值。
17 ; 程序从公共标签'start'开始。
18
19 .wrap_target
20 do_1:
21 nop side 0 [5] ; 低电平持续6个周期（5个延迟周期，加1个nop周期）
22 jmp get_bit side 1 [3] ; 高电平持续4个周期。'get_bit'需要额外2个周期。
```

```

23 do_0:
24 nop side 1 [5] ; 输出高电平持续6个周期
25 nop side 0 [3] ; 输出低电平，持续4个周期
26 public start:
27 get_bit:
28 out x, 1 ; 始终从OSR移出一位到X，这样我们可以
29 jmp !x do_0 ; 根据该位进行跳转。Autopull在OSR为空时会自动补充。
30 .wrap

```

从名为 `start` 的标签开始，该程序每次将一个数据位移入 X 寄存器，从而基于该值进行跳转。根据结果，它使用 side-set 驱动所选 GPIO 上的 1-0 或 0-1 序列。该程序使用 autopull（第 11.5.4.2 节）自动从 TX FIFO 补充 OSR，在移出一定数据量后，无需中断程序控制流程或时序。此功能由 `.pio` 文件中的辅助函数启用，该函数配置并启动状态机：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\\_encoding/manchester\\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio) 第 33 至 46 行

```

33 static inline void manchester_tx_program_init(PIO pio, uint sm, uint offset, uint pin, float
 div) {
34 pio_sm_set_pins_with_mask(pio, sm, 0, 1u << pin);
35 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, true);
36 pio_gpio_init(pio, pin);
37
38 pio_sm_config c = manchester_tx_program_get_default_config(offset);
39 sm_config_set_sideset_pins(&c, pin);
40 sm_config_set_out_shift(&c, true, true, 32);
41 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
42 sm_config_set_clkdiv(&c, div);
43 pio_sm_init(pio, sm, offset + manchester_tx_offset_start, &c);
44
45 pio_sm_set_enabled(pio, sm, true);
46 }

```

另一状态机可以被编程以从传输信号中恢复原始数据：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\\_encoding/manchester\\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio) 第 49 至 71 行

```

49 .program manchester_rx
50
51 ; 假设线路空闲为低电平，且第一位为0
52 ; 一个比特为12个周期
53 ; '0' 编码为10
54 ; '1' 编码为01
55 ;
56 ; IN基准和JMP引脚映射必须指向用于接收 (RX) 的GPIO。
57 ; 必须启用自动推送 (Autopush)。
58 ; 在启用状态机 (SM) 之前，应将其置于“wait 1, pin”状态，
59 ; 以确保其在初始线路空闲状态结束前不会开始采样。
60
61 start_of_0: ; 当前处于0的0.25比特时——信号为高电平
62 wait 0 pin 0 ; 等待1到0的转换——此时已进入比特的0.5处
63 在 y, 1 [8] ; 发出0，休眠3/4比特时间
64 jmp pin start_of_0; 若信号再次为1，则表示另一个0比特，否则为1比特
65
66 .wrap_target
67 start_of_1: ; 我们处于1的第0.25比特处——信号为1
68 等待1引脚为0 ; 等待0到1的转换——此时处于比特的第0.5位置
69 在 x 中, 1 [8] ; 发送一个1，暂停3/4比特时间
70 jmp pin start_of_0; 如果信号再次为0，则表示另一个1比特，否则为0

```

```
71 .wrap
```

此处主要难点在于保持与输入转换的同步，因发射器与接收器的时钟可能彼此漂移。在曼彻斯特编码中，符号中心处总有一次转换，依据初始线路状态（高电平或低电平），即可判断转换方向，从而可使用 `wait` 指令在每个数据比特处重新同步线路转换。

本程序要求X寄存器和Y寄存器分别初始化为1和0，以便为 `in` 指令提供常量1或0。用于配置状态机的代码通过执行一些 `set` 指令初始化这些寄存器，然后启动程序运行。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\\_encoding/manchester\\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio) 第74至94行

```
74 static inline void manchester_rx_program_init(PIO pio, uint sm, uint offset, uint pin, float
 div) {
75 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, false);
76 pio_gpio_init(pio, pin);
77
78 pio_sm_config c = manchester_rx_program_get_default_config(offset);
79 sm_config_set_in_pins(&c, pin); // 用于WAIT
80 sm_config_set_jmp_pin(&c, pin); // 用于JMP
81 sm_config_set_in_shift(&c, true, true, 32);
82 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_RX);
83 sm_config_set_clkdiv(&c, div);
84 pio_sm_init(pio, sm, offset, &c);
85
86 // X 和 Y 被设置为0和1，以便于向ISR/FIFO发送这两个值。
87 pio_sm_exec(pio, sm, pio_encode_set(pio_x, 1));
88 pio_sm_exec(pio, sm, pio_encode_set(pio_y, 0));
89 // 假设线路空闲为低电平，且首个传输位为0。将状态机置于
90 // 启用前的等待状态。一旦检测到第一个0符号，RX将开始
91 // 接收。
92 pio_sm_exec(pio, sm, pio_encode_wait_pin(1, 0) | pio_encode_delay(2));
93 pio_sm_set_enabled(pio, sm, true);
94 }
```

SDK中的示例C程序将从GPIO2以约10 Mb/s的速率向GPIO3传输曼彻斯特编码的串行数据，前提是假设系统时钟为125 MHz。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\\_encoding/manchester\\_encoding.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.c) 第20行至第43行

```
20 int main() {
21 stdio_init_all();
22
23 PIO pio = pio0;
24 uint sm_tx = 0;
25 uint sm_rx = 1;
26
27 uint offset_tx = pio_add_program(pio, &manchester_tx_program);
28 uint offset_rx = pio_add_program(pio, &manchester_rx_program);
29 printf("发送程序加载地址: %d\n", offset_tx);
30 printf("接收程序加载地址: %d\n", offset_rx);
31
32 manchester_tx_program_init(pio, sm_tx, offset_tx, pin_tx, 1.f);
33 manchester_rx_program_init(pio, sm_rx, offset_rx, pin_rx, 1.f);
34
35 pio_sm_set_enabled(pio, sm_tx, false);
36 pio_sm_put_blocking(pio, sm_tx, 0);
37 pio_sm_put_blocking(pio, sm_tx, 0x0ff0a55a);
38 pio_sm_put_blocking(pio, sm_tx, 0x12345678);
```

```

39 pio_sm_set_enabled(pio, sm_tx, true);
40
41 for (int i = 0; i < 3; ++i)
42 printf("%08x\n", pio_sm_get_blocking(pio, sm_rx));
43 }

```

### 11.6.6. 差分曼彻斯特编码（BMC）发送与接收

图60。差分曼彻斯特特串行线路编码，也称为双相标记编码（BMC）。

线路在每个

比特周期开始时发生跳变。

比特周期中间

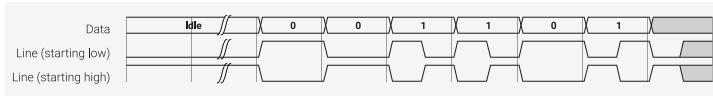
的跳变表示一个<sub>1</sub>数据位，

跳变的缺失表示

一个<sub>0</sub>位。无论

线路初始状态为高还是低，编码

规则均相同。



发送程序与曼彻斯特编码示例类似：它反复将位从OSR移入X寄存器（依赖自动拉取机制在后台填充OSR），执行分支，并根据该位的值驱动GPIO引脚电平上下跳变。

额外的复杂性在于，驱动引脚的信号模式不仅取决于数据位的值（如普通曼彻斯特编码），还取决于上一比特周期结束时线路的状态。这在图60中有所示，当线路初始为高电平时，模式被反转。为应对此情况，测试与驱动代码各有两份副本，分别对应每种初始线路状态，这些副本通过一系列跳转按正确顺序连接。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\\_manchester/differential\\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio) 第8至35行

```

8 .program differential_manchester_tx
9 .side_set 1 opt
10
11 ; 每16个周期传输一位。在每个位周期内：
12 ; - ‘0’ 编码为位周期开始时发生跳变
13 ; - ‘1’ 编码为位周期开始及中间均发生跳变
14 ;
15 ; Side-set位0必须映射至数据输出引脚。
16 ; 必须启用Autopull功能。
17
18 public start:
19 initial_high:
20 out x, 1 ; 位周期开始：始终断言跳变
21 jmp !x high_0 side 1 [6] ; 测试我们刚刚从只读存储器（OSR）移出的数据位
22 high_1:
23 nop
24 jmp initial_high side 0 [6] ; 对于`1`位，也在中间发生跳变
25 high_0:
26 jmp initial_low [7] ; 否则，线路在中间保持稳定状态
27
28 initial_low:
29 out x, 1 ; 始终从只读存储器（OSR）向X寄存器移位1位，以便我们能
30 jmp !x low_0 side 0 [6] ; 依据该位进行分支。自动拉取功能会为我们重新填充OSR。
31 low_1:
32 nop
33 jmp initial_low side 1 [6] ; 如果有两次跳变，则返回
34 low_0:
35 jmp initial_high [7] ; 初始线路状态已翻转！

```

该.pio文件还包含一个辅助函数，用于初始化差分曼彻斯特TX的状态机，并将其连接到指定GPIO。我们任意选择了32位帧大小和LSB优先串行化（shift\_right 在 sm\_config\_set\_out\_shift 中设置为 true），但由于程序每次处理一位，我们可以通  
过重新配置状态机来更改此设置。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\\_manchester/differential\\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio) 第38至53行

```

38 static inline void differential_manchester_tx_program_init(PIO pio, uint sm, uint offset,
 uint pin, float div) {
39 pio_sm_set_pins_with_mask(pio, sm, 0, 1u << pin);
40 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, true);
41 pio_gpio_init(pio, pin);
42
43 pio_sm_config c = differential_manchester_tx_program_get_default_config(offset);
44 sm_config_set_sideset_pins(&c, pin);
45 sm_config_set_out_shift(&c, true, true, 32);
46 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_TX);
47 sm_config_set_clkdiv(&c, div);
48 pio_sm_init(pio, sm, offset + differential_manchester_tx_offset_start, &c);
49
50 // 执行阻塞式拉取，以保持初始线路状态直至数据可用
51
52 pio_sm_exec(pio, sm, pio_encode_pull(false, true));
53 pio_sm_set_enabled(pio, sm, true);
54 }
```

接收程序采用以下策略：

1. 等待位周期开始时的初始跳变，以维持与发送时钟的对齐
2. 随后，等待已配置位周期的3/4，以使采样位于第二个半位周期的中心（参见图60）
3. 此时采样线路，以判定该位周期内存在一处或两处跳变
4. 重复

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\\_manchester/differential\\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio) 第55至85行

```

55 .program differential_manchester_rx
56
57 ; 假设线路空闲时为低电平
58 ; 一个比特周期为16个时钟周期。在每个比特周期内：
59 ; - '0' 通过在时间0处发生跳变进行编码
60 ; - '1' 通过在时间0和时间T/2处发生跳变进行编码
61 ;
62 ; IN映射和JMP引脚选择均必须映射至用于
63 ; 接收（RX）数据的GPIO引脚。必须启用自动推送（Autopush）。
64
65 public start:
66 initial_high: ; 在比特周期开始时检测上升沿
67 等待1号引脚, 0 [11]; 延时至半周期观察点（即3/4周期处）
68 跳转至 pin high_0 ; 比特周期）并根据RX引脚的高/低电平分支。
69 high_1:
70 输入x, 1 ; 检测到第二次跳变（数据符号`1`）
71 跳转至 initial_high
72 high_0:
73 输入y, 1 [1] ; 线路仍处于高电平，未发生中心跳变（数据为`0`）
74 ; 贯穿
75
76 .wrap_target
77 initial_low: ; 在位周期开始时检测下降沿
78 wait 0 pin, 0 [11]; 延时至半周期后半段的眼区
79 jmp pin low_1
80 low_0:
81 in y, 1 ; 线路仍处于低电平，未发生中心跳变（数据为`0`）
82 跳转至 initial_high
83 low_1: ; 检测到第二次跳变（数据为`1`）
84 in x, 1 [1]
```

85 .wrap

此代码假设X和Y的值分别为1和0。该假设由所包含的C辅助函数进行设置：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\\_manchester/differential\\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio) 第88至104行

```

88 static inline void differential_manchester_rx_program_init(PIO pio, uint sm, uint offset,
89 uint pin, float div) {
90 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, false);
91 pio_gpio_init(pio, pin);
92
93 pio_sm_config c = differential_manchester_rx_program_get_default_config(offset);
94 sm_config_set_in_pins(&c, pin); // 用于WAIT
95 sm_config_set_jmp_pin(&c, pin); // 用于JMP
96 sm_config_set_in_shift(&c, true, true, 32);
97 sm_config_set_fifo_join(&c, PIO_FIFO_JOIN_RX);
98 sm_config_set_clkdiv(&c, div);
99 pio_sm_init(pio, sm, offset, &c);
100
101 // X 和 Y 分别设置为 0 和 1，便于将其发出至 ISR/FIFO。
102 pio_sm_exec(pio, sm, pio_encode_set(pio_x, 1));
103 pio_sm_exec(pio, sm, pio_encode_set(pio_y, 0));
104 pio_sm_set_enabled(pio, sm, true);
105 }
```

所有必要组件现已就绪，可在两根 GPIO 之间通过线路实现串行数据回环。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\\_manchester/differential\\_manchester.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.c)

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符: BSD-3-Clause
5 */
6
7 #include <stdio.h>
8
9 #include "pico/stlib.h"
10 #include "hardware/pio.h"
11 #include "differential_manchester.pio.h"
12
13 // 差分串行发送/接收示例
14 // 需要将一根导线从 GPIO2 连接到 GPIO3
15
16 const uint pin_tx = 2;
17 const uint pin_rx = 3;
18
19 int main() {
20 stdio_init_all();
21
22 PIO pio = pio0;
23 uint sm_tx = 0;
24 uint sm_rx = 1;
25
26 uint offset_tx = pio_add_program(pio, &differential_manchester_tx_program);
27 uint offset_rx = pio_add_program(pio, &differential_manchester_rx_program);
28 printf("发送程序加载于%d\n", offset_tx);
29 printf("接收程序加载于%d\n", offset_rx);
30 }
```

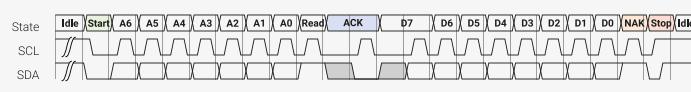
```

31 // 配置状态机，设置比特率为 5 Mbps
32 differential_manchester_tx_program_init(pio, sm_tx, offset_tx, pin_tx, 125.f / (16 * 5));
33 differential_manchester_rx_program_init(pio, sm_rx, offset_rx, pin_rx, 125.f / (16 * 5));
34
35 pio_sm_set_enabled(pio, sm_tx, false);
36 pio_sm_put_blocking(pio, sm_tx, 0);
37 pio_sm_put_blocking(pio, sm_tx, 0x0ff0a55a);
38 pio_sm_put_blocking(pio, sm_tx, 0x12345678);
39 pio_sm_set_enabled(pio, sm_tx, true);
40
41 for (int i = 0; i < 3; ++i)
42 printf("%08x\n", pio_sm_get_blocking(pio, sm_rx));
43 }

```

## 11.6.7. I2C

图61。一次1字节的I2C读取传输。在空闲状态下，两个线路均处于上拉悬空状态。主控端将SDA拉低（起始条件），随后发送7位地址A6-A0及一个方向位（读/写）。从控端将SDA拉低以确认地址（ACK），随后传输数据字节。从控端在SDA线上串行发送数据，由SCL时钟驱动。每第9个时钟周期，主控端将SDA拉低以确认数据，除最后一个字节外，该字节主控端保持SDA高电平（NAK）。在SCL处于高电平时释放SDA即构成停止条件，使总线返回空闲状态。



I2C是一种广泛应用的串行总线，最早描述于死海古卷，后由飞利浦半导体采用。该总线由两根带上拉电阻的线组成开放漏极结构，多个设备通过拉低或释放线路实现寻址及信号传输。它具有多项非同寻常的属性：

- SCL 可在任何时间、任何持续时间内，由总线上的任一成员保持为低电平（不一定是传输的目标或发起者）。此现象称为时钟延展（clock stretching）。总线在所有驱动器释放时钟之前不会前进。
- 总线成员既可作为一次传输的目标，又可发起其他传输（主/从角色并非固定不变）。然而，大多数 I2C 硬件对此支持较弱。
- SCL 非边沿敏感时钟，SDA 必须在 SCL 为高电平期间始终有效。
- 尽管 SDA 对 SCL 透明，但 SDA 在 SCL 高电平期间的跳变用于标示传输的开始与结束（即起始/停止），或同次传输中新地址阶段的开始（即重启）。

下述 PIO 程序在发起者角色中负责处理序列化、时钟延展及 ACK 检查。该程序提供一种机制，可在 FIFO 数据流中转义 PIO 指令，以便在适当时机发出起始/停止/重启序列。在未收到意外 NAK 的情况下，该功能可从 DMA 缓冲区执行长序列的 I2C 传输，无需处理器干预。

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio> 第8行至第73行

```

8 .program i2c
9 .side_set 1 opt pindirs
10
11 ; 发送编码：
12 ; | 15:10 | 9 | 8:1 | 0 |
13 ; | 指令 | 结束 | 数据 | NAK |
14 ;
15 ; 如果Instr的值n>0，则该FIFO字无
16 ; 数据负载，接下来的n+1个字节将作为指令执行。
17 ; 否则，先移出8位数据，然后移出ACK位。
18 ;
19 ; Instr机制允许处理器编程停止/启动/重复启动序列，
20 ; 并由状态机在数据流中的指定点执行，
21 ;
22 ;
23 ; "Final"字段应设置于传输的最后一个字节。
24 ; 此字段指示状态机忽略 NAK：若该字段未设置，
25 ; 任何 NAK 将导致状态机停止并产生中断。

```

```

26 ;
27 ; 应启用自动拉取功能，阈值设为 16。
28 ; 应启用自动推送功能，阈值设为 8。
29 ; 应使用半字写入访问 TX FIFO，以确保
30 ; 数据能立即在 OSR 中可用。
31 ;
32; 引脚映射:
33 ; - 输入引脚 0 为 SDA, 1 为 SCL (若使用时钟拉伸)
34; - 跳转引脚为 SDA
35; - 侧边设置引脚 0 为 SCL
36; - 设定引脚 0 为 SDA
37; - 输出引脚 0 为 SDA
38 ; - SCL 必须等于 SDA + 1 (用于等待映射)
39 ;
40 ; 系统 IO 控制中应对 OE 输出进行取反!
41 ; (在本程序中可执行取反,
42 ; 但将消耗 2 条指令：一条用于取反，另一条用于处理
43 ; 伴随 MOV 指令对 TX 移位计数器产生的副作用。
44
45 do_nack:
46 jmp y-- entry_point ; 如预期收到 NAK，则继续
47 irq wait 0 rel ; 否则停止，并寻求帮助
48
49 do_byte:
50 set x, 7 ; 循环8次
51 bitloop:
52 out pindirs, 1 [7] ; 序列化写入数据 (读取时全为 1)
53 nop side 1 [2] ; SCL 上升沿
54 wait 1 pin, 1 [4] ; 允许时钟拉伸
55 in pins, 1 [7] ; 在 SCL 脉冲中间采样读取的数据
56 jmp x-- bitloop side 0 [7] ; SCL 下降沿
57
58 ; 处理 ACK 脉冲
59 out pindirs, 1 [7] ; 读取时，我们提供ACK响应。
60 nop side 1 [7]; SCL 上升沿
61 wait 1 pin, 1 [7] ; 允许时钟拉伸
62 jmp 引脚 do_nack 侧 0 [2]; 测试 SDA 是否为 ACK／NAK，若为 ACK 则继续执行
63
64 公共入口点:
65 .wrap_target
66 out x, 6 ; 解包指令计数
67 out y, 1 ; 解包 NAK 忽略位
68 jmp !x do_byte ; 指令 == 0，此为数据记录。
69 out null, 32 ; 指令 > 0，此 OSR 余下部分无效
70 do_exec:
71 out exec, 16 ; 每个 FIFO 字执行一条指令
72 jmp x-- do_exec ; 重复 n+1 次
73 .wrap

```

I2C 程序所需的 IO 映射较为复杂，原因在于两条串行线路必须以不同方式进行驱动和采样。一个值得注意的特性是，状态机在输出为低电平时必须将输出使能置高，因总线采用开漏设计，数据的逻辑含义因此被反转。此项处理可在 PIO 程序中完成（例如 `mov osr, ~osr`），但我们也可利用 RP2350 的 IO 控制功能，在 GPIO 复用器中实现此反转，从而节省指令数。

Pico 示例地址：<https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio>，范围为第 81 行至第 121 行

```

81 static inline void i2c_program_init(PIO pio, uint sm, uint offset, uint pin_sda, uint
82 pin_scl) {
83 assert(pin_scl == pin_sda + 1);
84 pio_sm_config c = i2c_program_get_default_config(offset);

```

```

84
85 // IO 映射
86 sm_config_set_out_pins(&c, pin_sda, 1);
87 sm_config_set_set_pins(&c, pin_sda, 1);
88 sm_config_set_in_pins(&c, pin_sda);
89 sm_config_set_sideset_pins(&c, pin_scl);
90 sm_config_set_jmp_pin(&c, pin_sda);
91
92 sm_config_set_out_shift(&c, false, true, 16);
93 sm_config_set_in_shift(&c, false, true, 8);
94
95 float div = (float)clock_get_hz(clk_sys) / (32 * 100000);
96 sm_config_set_clkdiv(&c, div);
97
98 // 尽量避免在连接 IO 时引起总线毛刺。先将
99 // 设置为当 PIO 断言 OE 置低时引脚被拉低，并保持上拉
100 // 否则。
101 gpio_pull_up(pin_scl);
102 gpio_pull_up(pin_sda);
103 uint32_t both_pins = (1u << pin_sda) | (1u << pin_scl);
104 pio_sm_set_pins_with_mask(pio, sm, both_pins, both_pins);
105 pio_sm_set_pindirs_with_mask(pio, sm, both_pins, both_pins);
106 pio_gpio_init(pio, pin_sda);
107 gpio_set_oeover(pin_sda, GPIO_OVERRIDE_INVERT);
108 pio_gpio_init(pio, pin_scl);
109 gpio_set_oeover(pin_scl, GPIO_OVERRIDE_INVERT);
110 pio_sm_set_pins_with_mask(pio, sm, 0, both_pins);
111
112 // 启动前清除 IRQ 标志，并确保该标志不会实际
113 // 触发系统级中断（该标志仅作为状态标识）
114 pio_set_irq0_source_enabled(pio, (enum pio_interrupt_source) ((uint) pis_interrupt0 +
sm), false);
115 pio_set_irq1_source_enabled(pio, (enum pio_interrupt_source) ((uint) pis_interrupt0 +
sm), false);
116 pio_interrupt_clear(pio, sm);
117
118 // 配置并启动状态机
119 pio_sm_init(pio, sm, offset + i2c_offset_entry_point, &c);
120 pio_sm_set_enabled(pio, sm, true);
121 }

```

我们也可以使用PIO汇编器生成一张指令表，用于通过FIFO传递Start/Stop/Rstart条件。

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio> 第126至136行

```

126 .program set_scl_sda
127 .side_set 1 opt
128
129 ; 组装一张指令表，软件可以从中选择，并将130；传入FIFO，以发出START/STOP/RSTART信号。此
程序并非设计为完整程序运行。131；
130
131
132
133 set pindirs, 0 side 0 [7] ; SCL = 0, SDA = 0
134 set pindirs, 1 side 0 [7] ; SCL = 0, SDA = 1
135 set pindirs, 0 side 1 [7] ; SCL = 1, SDA = 0
136 set pindirs, 1 side 1 [7] ; SCL = 1, SDA = 1

```

示例代码在状态机的 FIFO 上执行阻塞式软件 IO，以避免配置系统 DMA 的额外复杂性。例如，I2C 启动条件的入队如下：

Pico 示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio\\_i2c.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio_i2c.c) 第 69 至 74 行

```

69 void pio_i2c_start(PIO pio, uint sm) {
70 pio_i2c_put_or_err(pio, sm, 2u << PIO_I2C_ICOUNT_LSB); // 转义
 3条指令序列的代码
71 pio_i2c_put_or_err(pio, sm, set_scl_sda_program_instructions[I2C_SC1_SD0]); // 我们已处于空闲状态
 , 直接将SDA拉低
72 pio_i2c_put_or_err(pio, sm, set_scl_sda_program_instructions[I2C_SC0_SD0]); // 同时将时钟拉低以
 便传输数据
73 pio_i2c_put_or_err(pio, sm, pio_encode_mov(pio_isr, pio_null)); // 确保
 写入操作后_ISR计数器已清零
74 }
```

由于I2C在多个环节可能出错，必须能够检查状态机的错误标志，清除暂停状态并重新启动，随后才断言停止条件并释放总线。

Pico 示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio\\_i2c.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio_i2c.c) 第15至17行

```

15 bool pio_i2c_check_error(PIO pio, uint sm) {
16 return pio_interrupt_get(pio, sm);
17 }
```

Pico 示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio\\_i2c.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio_i2c.c) 第19至23行

```

19 void pio_i2c_resume_after_error(PIO pio, uint sm) {
20 pio_sm_drain_tx_fifo(pio, sm);
21 pio_sm_exec(pio, sm, (pio->sm[sm].execctrl & PIO_SM0_EXECCTRL_WRAP_BOTTOM_BITS) >>
 PIO_SM0_EXECCTRL_WRAP_BOTTOM_LSB);
22 pio_interrupt_clear(pio, sm);
23 }
```

我们需要一些更高级的函数，能够通过FIFO传递格式正确的数据，并在适当位置插入起始、停止、NAK等信号。这足以  
为RP2350上的其他硬件I2C提供类似的接口。

Pico示例: [https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c\\_bus\\_scan.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c_bus_scan.c) 第13至42行

```

13 int main() {
14 stdio_init_all();
15
16 PIO pio = pio0;
17 uint sm = 0;
18 uint offset = pio_add_program(pio, &i2c_program);
19 i2c_program_init(pio, sm, offset, PIN_SDA, PIN_SCL);
20
21 printf("\nPIO I2C总线扫描\n");
22 printf(" 0 1 2 3 4 5 6 7 8 9 A B C D E F\n");
23
24 for (int addr = 0; addr < (1 << 7); ++addr) {
25 if (addr % 16 == 0) {
26 printf("%02x ", addr);
27 }
28 // 从探针地址执行 0 字节读取。读取函数
29 // 除最后一个数据字节外，任何时候都返回负结果 NAK。
30 // byte. 跳过保留地址。
31 int result;
32 if (reserved_addr(addr))
33 result = -1;
34 else
```

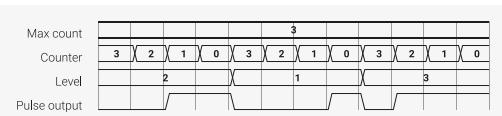
```

35 result = pio_i2c_read_blocking(pio, sm, addr, NULL, 0);
36
37 printf(result < 0 ? "." : "@");
38 printf(addr % 16 == 15 ? "\n" : " ");
39 }
40 printf("完成。\\n");
41 return 0;
42 }

```

### 11.6.8. PWM

图 62。脉宽调制（PWM）。状态机以固定时间间隔输出正电压脉冲。脉冲宽度受到控制，使线路在某一受控的时间比例内保持高电平（占空比）。此用法之一是通过以高于人类视觉暂留速度的频率脉冲控制LED，实现亮度的平滑调节。



该程序反复使用Y寄存器从指定数值倒计数至0，同时将Y寄存器中的计数值与存储于X寄存器的脉冲宽度进行比较。计数开始前，输出端保持低电平；当Y的值达到X时，输出端置为高电平。Y递减至0后，流程重复，输出端再次被拉低。因此，输出为高电平的时间比例与存储于X寄存器中的脉冲宽度成正比。

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.pio> 第10至22行

```

10 .program pwm
11 .side_set 1 opt
12
13 pull noblock side 0 ; 若 FIFO 中有数据，则拉取至 OSR；否则复制 X 至 OSR。
14 mov x, osr ; 将最近拉取的数据复制回寄存器 X
15 mov y, isr ; ISR 包含 PWM 周期。Y 用作计数器。
16 countloop:
17 jmp x!=y noset ; 若 X == Y，则将引脚置高，保持两条路径长度匹配。
18 jmp skip 侧 1
19 noset:
20 nop ; 单个空指令周期以保持两条路径长度一致。
21 skip:
22 jmp y-- countloop ; 循环直到 Y 变为 0，然后从 FIFO 拉取新的 PWM 值。

```

通常，PWM 可在特定脉冲宽度保持数千个脉冲，无需每次都提供新的脉冲宽度。本示例展示了非阻塞 PULL（第 11.4.7 节）如何实现此功能：若 TX FIFO 为空，非阻塞 PULL 会将 X 复制到 OSR。拉取后，程序将 OSR 复制回 X，以供与 Y 中的计数值比较。最终效果是，若在该周期开始时 TX FIFO 未提供新的占空比值，则会重复使用（通过失败的 PULL 将 X 复制到 OSR，再通过 MOV 复制回 X 的）上一个周期的占空比值，重复使用所需的周期数。

这里展示的另一种有用技术是在不需要 IN 指令时，将 ISR 用作配置寄存器。

系统软件可以向 ISR 加载任意 32 位数值（通过直接在状态机上执行指令），程序将在每次开始计数时将该数值复制到 Y 寄存器。ISR 可用于配置 PWM 计数范围、状态机的时钟分频器控制计数速率。

要开始调制脉冲，首先需将状态机的 side-set 引脚映射至欲输出 PWM 的 GPIO，并告知状态机程序在 PIO 指令存储器中的加载位置：

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.pio> 第25至31行

```

25 static inline void pwm_program_init(PIO pio, uint sm, uint offset, uint pin) {
26 pio_gpio_init(pio, pin);
27 pio_sm_set_consecutive_pindirs(pio, sm, pin, 1, true);

```

```

28 pio_sm_config c = pwm_program_get_default_config(offset);
29 sm_config_set_sideset_pins(&c, pin);
30 pio_sm_init(pio, sm, offset, &c);
31 }
```

加载ISR以设置所需计数范围需要一定的准备工作：

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c> 第14至20行

```

14 void pio_pwm_set_period(PIO pio, uint sm, uint32_t period) {
15 pio_sm_set_enabled(pio, sm, false);
16 pio_sm_put_blocking(pio, sm, period);
17 pio_sm_exec(pio, sm, pio_encode_pull(false, false));
18 pio_sm_exec(pio, sm, pio_encode_out(pio_isr, 32));
19 pio_sm_set_enabled(pio, sm, true);
20 }
```

一旦完成，状态机即可启用，并且 PWM 值可以直接写入其 TX FIFO。

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c> 第23-25行

```

23 void pio_pwm_set_level(PIO pio, uint sm, uint32_t level) {
24 pio_sm_put_blocking(pio, sm, level);
25 }
```

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c> 第27-51行

```

27 int main() {
28 stdio_init_all();
29 #ifndef PICO_DEFAULT_LED_PIN
30 #warning pio/pwm 示例需要带常规 LED 的开发板
31 puts("未定义默认 LED 引脚");
32 #else
33
34 // 待办 获取空闲的状态机
35 PIO pio = pio0;
36 int sm = 0;
37 uint offset = pio_add_program(pio, &pwm_program);
38 printf("Loaded program at %d\n", offset);
39
40 pwm_program_init(pio, sm, offset, PICO_DEFAULT_LED_PIN);
41 pio_pwm_set_period(pio, sm, (1u << 16) - 1);
42
43 int level = 0;
44 while (true) {
45 printf("Level = %d\n", level);
46 pio_pwm_set_level(pio, sm, level * level);
47 level = (level + 1) % 256;
48 sleep_ms(10);
49 }
50 #endif
51 }
```

若TX FIFO持续补充新脉冲宽度值，该程序将为每个脉冲消耗一个新的脉冲宽度。一旦FIFO耗尽，程序将重新使用最近提供的数值。

### 11.6.9. 加法运算

尽管未设计用于计算，PIO极有可能是图灵完备的，前提是在纸带足够长的情况下。据推测，若时钟频率足够高，它可能能够运行DOOM。

Pico示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/addition/addition.pio> 第7至25行

```

7 .program addition
8
9 ;从TX FIFO中弹出两个32位整数，将它们相加，并将结果（10）推回TX FIFO。由于我们使用显式的
10 ;推送和弹出指令，故应禁用自动推送/弹出功能。
11 ;显式的推送和弹出指令。
12 ;
13 ;该程序使用二进制补码恒等式 $x + y == \sim(\sim x - y)$
14
15 pull
16 mov x, ~osr
17 pull
18 mov y, osr
19 jmp test ;该循环等同于以下C语言代码：
20 incr: ; while (y--)
21 jmp x-- test ; x--;
22 test: ;该循环最终实现了从x中减去y的效果。
23 jmp y-- incr
24 mov isr, ~x
25 push

```

一个完整的32位加法在125 MHz时钟下仅需约一分钟。程序从TX FIFO中读取两个数字，并将它们的和写入RX FIFO，这非常适合与系统DMA协作，或由处理器直接操作：

Pico 示例：<https://github.com/raspberrypi/pico-examples/blob/master/pio/addition/addition.c>

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符 : BSD-3-Clause
5 */
6
7 #include <stdlib.h>
8 #include <stdio.h>
9
10 #include "pico/stl.h"
11 #include "hardware/pio.h"
12 #include "addition.pio.h"
13
14 // 快问快答：调用此函数时处理器执行了多少次加法？
15 uint32_t do_addition(PIO pio, uint sm, uint32_t a, uint32_t b) {
16 pio_sm_put_blocking(pio, sm, a);
17 pio_sm_put_blocking(pio, sm, b);
18 return pio_sm_get_blocking(pio, sm);
19 }
20
21 int main() {
22 stdio_init_all();
23
24 PIO pio = pio0;
25 uint sm = 0;
26 uint offset = pio_add_program(pio, &addition_program);
27 addition_program_init(pio, sm, offset);
28
29 printf("执行一些随机加法操作:\n");

```

```

30 for (int i = 0; i < 10; ++i) {
31 uint a = rand() % 100;
32 uint b = rand() % 100;
33 printf("%u + %u = %u\n", a, b, do_addition(pio, sm, a, b));
34 }
35 }
```

### 11.6.10. 进一步示例

Raspberry Pi Pico系列C/C++ SDK包含一个PIO章节，详细介绍了此处未涉及的部分软件相关主题。其中包括一个PIO + DMA逻辑分析仪示例，能够在每个周期采样所有GPIO（在125 MHz时带宽接近4Gbps，但这会较快耗尽RP2350的RAM）。

在Pico示例库的 `pio/` 目录中还有更多示例。

部分更具实验性质的示例代码，如DPI和SD卡支持，当前存放于Pico Extras与Pico Playground代码库中。这些代码中的PIO部分功能完整，但相关软件栈仍处于实验状态。

## 11.7. 寄存器列表

PIO0和PIO1寄存器的基地址分别为 `0x50200000` 和 `0x50300000`（在SDK中定义为PIO0\_BASE和PIO1\_BASE）。

表981. PIO  
寄存器列表

偏移量	名称	说明
0x000	<code>CTRL</code>	PIO控制寄存器
0x004	<code>FSTAT</code>	FIFO状态寄存器
0x008	<code>FDEBUG</code>	FIFO调试寄存器
0x00c	<code>FLEVEL</code>	FIFO水位
0x010	<code>TXF0</code>	此状态机的TX FIFO的直接写访问端口。每次写入操作将向FIFO推送一个字。写入已满的FIFO不会改变FIFO状态或内容，并会为该FIFO设置粘性FDEBUG_TXOVER错误标志。
0x014	<code>TXF1</code>	此状态机的TX FIFO的直接写访问端口。每次写入操作将向FIFO推送一个字。写入已满的FIFO不会改变FIFO状态或内容，并会为该FIFO设置粘性FDEBUG_TXOVER错误标志。
0x018	<code>TXF2</code>	此状态机的TX FIFO的直接写访问端口。每次写入操作将向FIFO推送一个字。写入已满的FIFO不会改变FIFO状态或内容，并会为该FIFO设置粘性FDEBUG_TXOVER错误标志。
0x01c	<code>TXF3</code>	此状态机的TX FIFO的直接写访问端口。每次写入操作将向FIFO推送一个字。写入已满的FIFO不会改变FIFO状态或内容，并会为该FIFO设置粘性FDEBUG_TXOVER错误标志。

偏移量	名称	说明
0x020	RXF0	对该状态机的RX FIFO进行直接读取访问。每次读取会从FIFO弹出一个字。尝试从空FIFO读取不会影响FIFO状态，但会为该FIFO设置粘滞的FDEBUG_RXUNDER错误标志。从空FIFO读取返回的数据未定义。
0x024	RXF1	对该状态机的RX FIFO进行直接读取访问。每次读取会从FIFO弹出一个字。尝试从空FIFO读取不会影响FIFO状态，但会为该FIFO设置粘滞的FDEBUG_RXUNDER错误标志。从空FIFO读取返回的数据未定义。
0x028	RXF2	对该状态机的RX FIFO进行直接读取访问。每次读取会从FIFO弹出一个字。尝试从空FIFO读取不会影响FIFO状态，但会为该FIFO设置粘滞的FDEBUG_RXUNDER错误标志。从空FIFO读取返回的数据未定义。
0x02c	RXF3	对该状态机的RX FIFO进行直接读取访问。每次读取会从FIFO弹出一个字。尝试从空FIFO读取不会影响FIFO状态，但会为该FIFO设置粘滞的FDEBUG_RXUNDER错误标志。从空FIFO读取返回的数据未定义。
0x030	IRQ	状态机IRQ标志寄存器。写入1以清除。共有八个状态机IRQ标志，状态机可设置、清除并等待这些标志。标志与状态机之间无固定对应关系——任何状态机均可使用任一标志。  这八个标志中的任意一个均可用于状态机间的时序同步，结合IRQ和WAIT指令使用。这八个标志的任意组合也可与FIFO状态中断一起路由至两个系统级中断请求中的任意一个——详见如IRQ0_INTE。
0x034	IRQ_FORCE	向这些位写入1会强制激活对应IRQ。请注意，这与INTF寄存器不同： 此处写入会影响PIO内部状态。INTF仅用于断言面向处理器的IRQ信号以测试ISR，对状态机不可见。
0x038	INPUT_SYNC_BYPASS	每个GPIO输入端设有两级触发器同步器，以保护PIO逻辑免受亚稳态影响。这会增加输入延迟，对于高速同步IO（例如SPI），可能需要绕过这些同步器。该寄存器的每个位对应一个GPIO。  0 → 输入同步（默认） 1 → 同步器已绕过 若不确定，请将此寄存器保持全零。
0x03c	DBG_PADOUT	读取此寄存器以采样PIO当前驱动至GPIO的端口输出值。RP2040上共有30个GPIO，因此最高两位硬连线为0。
0x040	DBG_PADOE	读取此寄存器以采样PIO当前驱动至GPIO的端口输出使能（方向）。RP2040上共有30个GPIO，因此最高两位硬连线为0。

偏移量	名称	说明
0x044	DBG_CFGINFO	PIO 硬件包含若干在不同芯片产品间可能变化的自由参数。 这些参数应在芯片数据手册中予以说明，且此处亦有展示。
0x048	INSTR_MEM0	对指令存储器位置 0 的写入专用访问权限
0x04c	INSTR_MEM1	对指令存储器位置 1 的写入专用访问权限
0x050	INSTR_MEM2	对指令存储器位置 2 的写入专用访问权限
0x054	INSTR_MEM3	对指令存储器位置 3 的写入专用访问权限
0x058	INSTR_MEM4	对指令存储器位置 4 的写入专用访问权限
0x05c	INSTR_MEM5	对指令存储器位置 5 的写入专用访问权限
0x060	INSTR_MEM6	对指令存储器位置 6 的写入专用访问权限
0x064	INSTR_MEM7	对指令存储器位置 7 的写入专用访问权限
0x068	INSTR_MEM8	对指令存储器位置 8 的写入专用访问权限
0x06c	INSTR_MEM9	对指令存储器位置 9 的写入专用访问权限
0x070	INSTR_MEM10	对指令存储器位置 10 的写入专用访问权限
0x074	INSTR_MEM11	对指令存储器位置 11 的写入专用访问权限
0x078	INSTR_MEM12	对指令存储器位置 12 的写入专用访问权限
0x07c	INSTR_MEM13	对指令存储器位置 13 的写入专用访问权限
0x080	INSTR_MEM14	对指令存储器位置 14 的写入专用访问权限
0x084	INSTR_MEM15	对指令存储器位置 15 的写入专用访问权限
0x088	INSTR_MEM16	对指令存储器位置 16 的写入专用访问权限
0x08c	INSTR_MEM17	对指令存储器位置 17 的写入专用访问权限
0x090	INSTR_MEM18	对指令存储器位置 18 的写入专用访问权限
0x094	INSTR_MEM19	对指令存储器位置 19 的写入专用访问权限
0x098	INSTR_MEM20	对指令存储器位置 20 的写入专用访问权限
0x09c	INSTR_MEM21	对指令存储器位置 21 的写入专用访问权限
0x0a0	INSTR_MEM22	对指令存储器位置 22 的写入专用访问权限
0x0a4	INSTR_MEM23	对指令存储器位置 23 的写入专用访问权限
0x0a8	INSTR_MEM24	对指令存储器位置 24 的写入专用访问权限
0x0ac	INSTR_MEM25	对指令存储器位置 25 的写入专用访问权限
0x0b0	INSTR_MEM26	对指令存储器位置 26 的写入专用访问权限
0x0b4	INSTR_MEM27	对指令存储器位置 27 的写入专用访问权限
0x0b8	INSTR_MEM28	对指令存储器位置 28 的写入专用访问权限
0x0bc	INSTR_MEM29	对指令存储器位置 29 的写入专用访问权限
0x0c0	INSTR_MEM30	对指令存储器位置 30 的写入专用访问权限
0x0c4	INSTR_MEM31	对指令存储器位置 31 的写入专用访问权限

偏移量	名称	说明
0x0c8	<a href="#">SM0_CLKDIV</a>	状态机0的时钟分频寄存器 频率 = 时钟频率 / (CLKDIV_INT + CLKDIV_FRAC / 256)
0x0cc	<a href="#">SM0_EXECCTRL</a>	状态机0的执行与行为设置
0x0d0	<a href="#">SM0_SHIFTCTRL</a>	控制状态机0输入/输出移位寄存器的行为
0x0d4	<a href="#">SM0_ADDR</a>	状态机0当前指令地址
0x0d8	<a href="#">SM0_INSTR</a>	读取以查看状态机当前指令地址所指令令 0号程序计数器 写入以立即执行指令（包括跳转） 随后恢复执行。
0x0dc	<a href="#">SM0_PINCTRL</a>	状态机引脚控制
0x0e0	<a href="#">SM1_CLKDIV</a>	状态机1的时钟分频寄存器 频率 = 时钟频率 / (CLKDIV_INT + CLKDIV_FRAC / 256)
0x0e4	<a href="#">SM1_EXECCTRL</a>	状态机1的执行与行为设置
0x0e8	<a href="#">SM1_SHIFTCTRL</a>	控制状态机1输入/输出移位寄存器的行为
0x0ec	<a href="#">SM1_ADDR</a>	状态机1当前指令地址
0x0f0	<a href="#">SM1_INSTR</a>	读取以查看状态机1程序计数器当前指令地址所指令令 写入以立即执行指令（包括跳转），随后恢复执行。
0x0f4	<a href="#">SM1_PINCTRL</a>	状态机引脚控制
0x0f8	<a href="#">SM2_CLKDIV</a>	状态机2的时钟分频寄存器 频率 = 时钟频率 / (CLKDIV_INT + CLKDIV_FRAC / 256)
0x0fc	<a href="#">SM2_EXECCTRL</a>	状态机2的执行与行为设置
0x100	<a href="#">SM2_SHIFTCTRL</a>	控制状态机2输入/输出移位寄存器的行为
0x104	<a href="#">SM2_ADDR</a>	状态机2当前指令地址
0x108	<a href="#">SM2_INSTR</a>	读取以查看状态机当前指令地址所指令令 2的程序计数器 写入以立即执行指令（包括跳转），随后恢复执行。
0x10c	<a href="#">SM2_PINCTRL</a>	状态机引脚控制
0x110	<a href="#">SM3_CLKDIV</a>	状态机3的时钟分频寄存器 频率 = 时钟频率 / (CLKDIV_INT + CLKDIV_FRAC / 256)
0x114	<a href="#">SM3_EXECCTRL</a>	状态机3的执行与行为设置
0x118	<a href="#">SM3_SHIFTCTRL</a>	控制状态机3的输入/输出移位寄存器的行为
0x11c	<a href="#">SM3_ADDR</a>	状态机3的当前指令地址
0x120	<a href="#">SM3_INSTR</a>	读取以查看状态机3程序计数器当前指向的指令 写入以立即执行指令（包括跳转），随后恢复执行。

偏移量	名称	说明
0x124	SM3_PINCTRL	状态机引脚控制
0x128	RXF0_PUTGET0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM0的RX FIFO第0条目
0x12c	RXF0_PUTGET1	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM0的RX FIFO第1条目
0x130	RXF0_PUTGET2	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM0的RX FIFO第2条目
0x134	RXF0_PUTGET3	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM0的RX FIFO第3条目
0x138	RXF1_PUTGET0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM1的RX FIFO第0条目
0x13c	RXF1_PUTGET1	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM1的RX FIFO条目1。
0x140	RXF1_PUTGET2	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM1的RX FIFO条目2。
0x144	RXF1_PUTGET3	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM1的RX FIFO条目3。
0x148	RXF2_PUTGET0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM2的RX FIFO条目0。
0x14c	RXF2_PUTGET1	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM2的RX FIFO条目1。
0x150	RXF2_PUTGET2	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM2的RX FIFO条目2。
0x154	RXF2_PUTGET3	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM2的RX FIFO条目3。
0x158	RXF3_PUTGET0	当设置了SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM3的RX FIFO条目0。
0x15c	RXF3_PUTGET1	当设置了SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM3的RX FIFO条目1。
0x160	RXF3_PUTGET2	当设置了SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET时，可直接读写SM3的RX FIFO条目2。

偏移量	名称	说明
0x164	RXF3_PUTGET3	当设置了 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET 时，可直接读写 SM3 的 RX FIFO 条目 3。
0x168	GPIOBASE	将系统 GPIO 编号中的 GPIO 0（从 PIO 视角）重新定位，以支持 PIO 访问超过 32 个 GPIO。 仅支持值 0 和 16（仅可写第 4 位）。
0x16c	中断	原始中断
0x170	IRQ0_INTE	irq0 中断使能
0x174	IRQ0_INTF	irq0 中断强制
0x178	IRQ0_INTS	irq0 掩码及强制后的中断状态
0x17c	IRQ1_INTE	用于 irq1 的中断使能
0x180	IRQ1_INTF	用于 irq1 的中断强制
0x184	IRQ1_INTS	irq1 掩码和强制后的中断状态

## PIO: CTRL 寄存器

偏移: 0x000

### 描述

PIO 控制寄存器

表 982. CTRL  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>NEXTPREV_CLKDIV_RESTART:</b> 写入 1 以重启相邻 PIO 块中状态机的时钟分频器，相邻关系由同一写入操作中的 NEXT_PIO_MASK 和 PREV_PIO_MASK 指定。  这相当于向这些 PIO 的 CTRL 寄存器中对应的 CLKDIV_RESTART 位写入 1。	SC	0x0
25	<b>NEXTPREV_SM_DISABLE:</b> 写入 1 以禁用相邻 PIO 块中的状态机，相邻关系由同一写入操作中的 NEXT_PIO_MASK 和 PREV_PIO_MASK 指定。  这相当于清除这些 PIO 的 CTRL 寄存器中对应的 SM_ENABLE 位。	SC	0x0
24	<b>NEXTPREV_SM_ENABLE:</b> 写入 1 以启用相邻 PIO 块中的状态机，相邻关系由同一写入操作中的 NEXT_PIO_MASK 和 PREV_PIO_MASK 指定。  这相当于在这些 PIO 的 CTRL 寄存器中设置相应的 SM_ENABLE 位。  如果同时设置了 OTHERS_SM_ENABLE 和 OTHERS_SM_DISABLE，则以禁用为优先。	SC	0x0

位	描述	类型	复位
23:20	<p><b>NEXT_PIO_MASK:</b> 系统中邻近、编号更高的PIO块（如果当前为最高编号PIO块，则为PIO块0）中状态机的掩码，用于在同一次写操作中应用NEXTPREV_CLKDIV_RESTART、NEXTPREV_SM_ENABLE和NEXTPREV_SM_DISABLE所指定的操作。</p> <p>该机制允许邻近PIO块中的状态机在写入此PIO块的CTRL寄存器时，实现启动、停止及时钟同步的完全同步。</p> <p>需注意，含有两个PIO的系统中，NEXT_PIO_MASK和PREV_PIO_MASK实际上指向同一PIO块。在此情况下，效果以累积方式施加（等同于对这些掩码进行按位或操作）。</p> <p>当一个PIO块允许非安全代码访问而另一个不允许时，邻近PIO块会被断开（状态信号被拉低至0，且控制信号被忽略）。</p>	SC	0x0
19:16	<p><b>PREV_PIO_MASK:</b> 系统中相邻编号较低的PIO块（若当前为PIO块0，则指编号最高的PIO块）中的状态机掩码，用以在同一次写入操作中对由OP_CLKDIV_RESTART、OP_ENABLE及OP_DISABLE指定的操作施加影响。</p> <p>该机制允许邻近PIO块中的状态机在写入此PIO块的CTRL寄存器时，实现启动、停止及时钟同步的完全同步。</p> <p>当一个PIO块允许非安全代码访问而另一个不允许时，邻近PIO块会被断开（状态信号被拉低至0，且控制信号被忽略）。</p>	SC	0x0
15:12	保留。	-	-
11:8	<p><b>CLKDIV_RESTART:</b> 从初始相位0重新启动状态机的时钟分频器。时钟分频器为自由运行状态，启动后其输出（包括分数抖动）完全由SMx_CLKDIV中配置的整数/分数除数决定。由此可知，若通过向该字段写入多个‘1’位同时重启多个除数相同的时钟分频器，则这些状态机的执行时钟将实现精确锁步。</p> <p>请注意，设置或清除SM_ENABLE不会停止时钟分频器的运行，因此一旦多个状态机的时钟同步，即便禁用或重新启用某一状态机，只要保持时钟分频器同步仍是安全的。</p> <p>另请注意，CLKDIV_RESTART 可在状态机运行时写入，这对于在分频器（SMx_CLKDIV）动态更改后重新同步时钟分频器非常有用。</p>	SC	0x0

位	描述	类型	复位
7:4	<p><b>SM_RESTART</b>: 写入1以立即清除内部状态机状态，该状态可能难以访问且将影响未来的执行。</p> <p>具体而言，将清除以下内容：输入和输出移位计数器；输入移位寄存器的内容；延迟计数器；等待IRQ的状态；写入SMx_INSTR或通过OUT/MOV EXEC执行的任何被阻塞指令；任何因OUT_STICKY而持续保持激活的引脚写操作。</p> <p>输出移位寄存器及X/Y暂存寄存器的内容不受影响。</p>	SC	0x0
3:0	<p><b>SM_ENABLE</b>: 通过对这四个位分别写入1或0来启用或禁用四个状态机中的每一个。禁用时，状态机将停止执行指令，除非该指令由系统直接写入SMx_INSTR。</p> <p>可以同时设置或清除多个位，以实现多个状态机的并行运行或停止。</p>	读写	0x0

## PIO: FSTAT 寄存器

偏移: 0x004

### 说明

FIFO 状态寄存器

表 983。FSTAT 寄存器

位	描述	类型	复位
31:28	保留。	-	-
27:24	<b>TXEMPTY</b> : 状态机 TX FIFO 为空	只读	0xf
23:20	保留。	-	-
19:16	<b>TXFULL</b> : 状态机 TX FIFO 已满	只读	0x0
15:12	保留。	-	-
11:8	<b>RXEMPTY</b> : 状态机 RX FIFO 为空	只读	0xf
7:4	保留。	-	-
3:0	<b>RXFULL</b> : 状态机 RX FIFO 已满	只读	0x0

## PIO: FDEBUG 寄存器

偏移: 0x008

### 说明

FIFO 调试寄存器

表 984。FDEBUG 寄存器

位	描述	类型	复位
31:28	保留。	-	-
27:24	<b>TXSTALL</b> : 状态机在阻塞 PULL 操作期间因 TX FIFO 为空，或在启用自动拉取的 OUT 操作中而停止。写入 1 以清除。	WC	0x0
23:20	保留。	-	-

位	描述	类型	复位
19:16	<b>TXOVER:</b> 出现 TX FIFO 溢出（即系统尝试写入满缓冲区）。写入 1 以清除。注意，写入满操作不会改变 FIFO 的状态或内容，但系统尝试写入的数据将被丢弃，因此若此标志被置位，您的软件极有可能丢失部分数据。	WC	0x0
15:12	保留。	-	-
11:8	<b>RXUNDER:</b> 发生了 RX FIFO 欠载（即系统空读取）。写入 1 以清除。请注意，空读取不会以任何方式改变 FIFO 的状态，但从空 FIFO 读取的数据是未定义的，因此该标志通常仅因某种软件错误而被置位。	WC	0x0
7:4	保留。	-	-
3:0	<b>RXSTALL:</b> 状态机在阻塞 PUSH 操作中因 RX FIFO 满而停滞，或在启用自动推送的 IN 操作时停滞。当对满 FIFO 进行非阻塞 PUSH 操作时，该标志也会被置位，此时状态机已丢弃数据。写入 1 以清除。	WC	0x0

## PIO: FLEVEL 寄存器

偏移: 0x00c

### 描述

FIFO 深度

表 985. FLEVEL  
寄存器

位	描述	类型	复位
31:28	<b>RX3</b>	只读	0x0
27:24	<b>TX3</b>	只读	0x0
23:20	<b>RX2</b>	只读	0x0
19:16	<b>TX2</b>	只读	0x0
15:12	<b>RX1</b>	只读	0x0
11:8	<b>TX1</b>	只读	0x0
7:4	<b>RX0</b>	只读	0x0
3:0	<b>TX0</b>	只读	0x0

## PIO: TXF0、TXF1、TXF2、TXF3 寄存器

偏移: 0x010, 0x014, 0x018, 0x01c

表 986. TXF0,  
TXF1, TXF2, TXF3  
寄存器

位	描述	类型	复位
31:0	此状态机的 TX FIFO 的直接写访问端口。每次写入操作将向 FIFO 推送一个字。写入已满的 FIFO 不会改变 FIFO 状态或内容，并会为该 FIFO 设置粘性 FDEB UG_TXOVER 错误标志。	WF	0x00000000

## PIO: RXF0、RXF1、RXF2、RXF3 寄存器

偏移: 0x020, 0x024, 0x028, 0x02c

表987。RXF0,  
RXF1, RXF2, RXF3  
寄存器

位	描述	类型	复位
31:0	对该状态机的RX FIFO进行直接读取访问。每次读取将从FIFO弹出一个字。尝试从空的FIFO读取不会改变FIFO的状态，但会为该FIFO设置粘滞的FDEBUG_RXUNDER错误标志。 从空的FIFO读取返回给系统的数据是未定义的。	RF	-

## PIO：IRQ寄存器

偏移量：0x030

表988。IRQ  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	状态机IRQ标志寄存器。写入1以清除。共有八个状态机IRQ标志，状态机可设置、清除并等待这些标志。标志与状态机之间无固定对应关系——任何状态机均可使用任一标志。  这八个标志中的任意一个均可用于状态机间的时序同步，结合IRQ和WAIT指令使用。这八个标志的任意组合也可与FIFO状态中断一起路由至两个系统级中断请求中的任意一个——详见如IRQ0_INTE。	WC	0x00

## PIO：IRQ\_FORCE寄存器

偏移：0x034

表989。IRQ\_FORCE  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	向这些位写入1将强制触发对应的IRQ。 注意，此处不同于INTF寄存器：在此写入会影响PIO的内部状态。INTF仅用于断言面向处理器的IRQ信号以测试ISR，对状态机不可见。	WF	0x00

## PIO：INPUT\_SYNC\_BYPASS寄存器

偏移量：0x038

表990。  
INPUT\_SYNC\_BYPASS  
寄存器

位	描述	类型	复位
31:0	每个GPIO输入端设有两级触发器同步器，以保护PIO逻辑免受亚稳态影响。这会增加输入延迟，对于高速同步IO（例如SPI），可能需要绕过这些同步器。 该寄存器的每个位对应一个GPIO。  0 → 输入同步（默认） 1 → 同步器已绕过 若不确定，请将此寄存器保持全零。	读写	0x00000000

## PIO：DBG\_PADOUT寄存器

偏移量：0x03c

表991。  
DBG\_PADOUT寄存器

位	描述	类型	复位
31:0	读取此寄存器以采样PIO当前驱动至GPIO的端口输出值。RP2040上共有30个G PIO，因此最高两位硬连线为0。	只读	0x00000000

## PIO: DBG\_PADOUT寄存器

偏移量: 0x040

表992。  
DBG\_PADOE寄存器

位	描述	类型	复位
31:0	读取此寄存器以采样PIO当前驱动至GPIO的端口输出使能（方向）。RP2040上共有30个GPIO，因此最高两位硬连线为0。	只读	0x00000000

## PIO: DBG\_CFGINFO寄存器

偏移量: 0x044

### 说明

PIO硬件具有一些可能因芯片型号而异的自由参数。

这些参数应在芯片数据手册中予以说明，且此处亦有展示。

表993。  
DBG\_CFGINFO  
寄存器

位	描述	类型	复位
31:28	<b>VERSION</b> : 核心PIO硬件版本。	只读	0x1
	枚举值：		
	0x0 → V0: 版本0 (RP2040)		
	0x1 → V1: 版本1 (RP2350)		
27:22	保留。	-	-
21:16	<b>IMEM_SIZE</b> : 指令存储器大小，单位为单条指令	只读	-
15:12	保留。	-	-
11:8	<b>SM_COUNT</b> : 此PIO实例所含状态机数量。	只读	-
7:6	保留。	-	-
5:0	<b>FIFO_DEPTH</b> : 状态机TX/RX FIFO的深度，单位为字。  通过SHIFTCTRL_FJOIN连接FIFO可形成一个深度为原来两倍的FIFO。	只读	-

## PIO: INSTR\_MEM0、INSTR\_MEM1、...、INSTR\_MEM30、INSTR\_MEM31寄存器

偏移量: 0x048、0x04c、...、0x0c0、0x0c4

表994。  
 $INSTR\_MEM0$   
 $\wedge INSTR\_MEM1$   
 $\wedge \dots \wedge INSTR\_M$   
 $EM30, INSTR\_MEM31$  寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	仅可写访问的指令存储位置 $N$	WO	0x0000

## PIO: SM0\_CLKDIV、SM1\_CLKDIV、SM2\_CLKDIV、SM3\_CLKDIV 寄存器

偏移量: 0x0c8、0x0e0、0x0f8、0x110

### 描述

状态机  $N$  的时钟分频寄存器

频率 = 时钟频率 / (CLKDIV\_INT + CLKDIV\_FRAC / 256)

表995。  
 $SM0\_CLKDIV$ ,  
 $SM1\_CLKDIV$ ,  
 $SM2\_CLKDIV$ ,  
 $SM3\_CLKDIV$  寄存器

位	描述	类型	复位
31:16	<b>INT:</b> 有效频率为 sysclk/(int + frac/256)。 值为0时，表示为65536。若 INT 为0，则 FRAC 也必须为0。	读写	0x0001
15:8	<b>FRAC:</b> 时钟分频器的分数部分	读写	0x00
7:0	保留。	-	-

## PIO: SM0\_EXECCTRL, SM1\_EXECCTRL, SM2\_EXECCTRL, SM3\_EXECCTRL 寄存器

偏移地址: 0x0cc, 0x0e4, 0x0fc, 0x114

### 描述

状态机的执行/行为设置  $N$

表996。  
 $SM0\_EXECCTRL$ ,  
 $SM1\_EXECCTRL$ ,  
 $SM2\_EXECCTRL$ ,  
 $SM3\_EXECCTRL$  寄存器

位	描述	类型	复位
31	<b>EXEC_STALLED:</b> 若为1，写入 $SMx\_INSTR$ 的指令将被阻塞，并由状态机锁存。指令完成后，将自动清零为0。	只读	0x0
30	<b>SIDE_EN:</b> 如果为1，延迟/侧边设置指令字段的最高有效位 (MSB) 将作为侧边设置启用位，而非侧边设置数据位。这允许指令可选择执行侧边设置，而非每条指令均执行，然而最大侧边设置宽度会由5位减少至4位。请注意，PINCTRL_SIDESET_COUNT 的数值包含此启用位。	读写	0x0
29	<b>SIDE_PINDIR:</b> 如果为1，侧边设置数据将用作引脚方向信号，而非引脚数值。	读写	0x0
28:24	<b>JMP_PIN:</b> 用作 JMP PIN 条件的 GPIO 编号，且不受输入映射影响。	读写	0x00
23:19	<b>OUT_EN_SEL:</b> 指定用于内联 OUT 使能的数据位。	读写	0x00
18	<b>INLINE_OUT_EN:</b> 如果为1，使用 OUT 数据位中的一位作为辅助写入使能。与 OUT_STICKY 结合使用时，使能为0的写入操作将撤销最近一次的引脚写入。此机制可实现有效的掩码及覆盖行为。  上述行为源于状态机引脚写入的优先级顺序 ( $SM0 < SM1 < \dots$ )。	读写	0x0
17	<b>OUT_STICKY:</b> 持续断言最近的 OUT/SET 信号至引脚	读写	0x0
16:12	<b>WRAP_TOP:</b> 达到该地址后，执行将回绕至 wrap_bottom。  若指令为跳转且跳转条件满足，则跳转操作优先执行。	读写	0x1f

位	描述	类型	复位
11:7	<b>WRAP_BOTTOM</b> : 达到 wrap_top 后，执行将回绕至该地址。	读写	0x00
6:5	<b>STATUS_SEL</b> : 用于 MOV x, STATUS 指令的比较选择。	读写	0x0
	枚举值：		
	0x0 → TXLEVEL: 当 TX FIFO 水平低于 N 时全为1，否则全为0		
	0x1 → RXLEVEL: 当 RX FIFO 水平低于 N 时全为1，否则全为0		
	0x2 → IRQ: 当索引的 IRQ 标志被触发时全为1，否则全为0		
4:0	<b>STATUS_N</b> : 用于 MOV x, STATUS 指令的比较级别或 IRQ 索引。 若 STATUS_SEL 为 TXLEVEL 或 RXLEVEL，则 STATUS_N 大于当前 FIFO 深度的值视为保留，行为未定义。	读写	0x00
	枚举值：		
	0x00 → IRQ: 此PIO块中IRQ标志的索引0-7		
	0x08 → IRQ_PREVPIO: 编号较低的PIO块中IRQ标志的索引0-7		
	0x10 → IRQ_NEXTPIO: 编号较高的PIO块中IRQ标志的索引0-7		

## PIO: SM0\_SHIFTCTRL, SM1\_SHIFTCTRL, SM2\_SHIFTCTRL, SM3\_SHIFTCTRL 寄存器

偏移: 0x0d0, 0x0e8, 0x100, 0x118

### 描述

控制状态机N输入/输出移位寄存器的行为

表997。  
SM0\_SHIFTCTRL,  
SM1\_SHIFTCTRL,  
SM2\_SHIFTCTRL,  
SM3\_SHIFTCTRL  
寄存器

位	描述	类型	复位
31	<b>FJOIN_RX</b> : 当值为1时，RX FIFO将占用TX FIFO的存储空间，深度加倍。  因此TX FIFO被禁用（始终显示为既满又空）。 当此位发生改变时，FIFO将被刷新。	读写	0x0
30	<b>FJOIN_TX</b> : 当该位为1时，TX FIFO将占用RX FIFO的存储空间，其深度加倍。  RX FIFO因此被禁用（其状态始终同时被读为满和空）。 当此位发生改变时，FIFO将被刷新。	读写	0x0
29:25	<b>PULL_THRESH</b> : 在自动拉取或条件拉取（PULL IFEMPTY）发生之前，从OSR移出的位数。 写入值32时实际写入0。	读写	0x00
24:20	<b>PUSH_THRESH</b> : 在自动推送或条件推送（PUSH IFFULL）发生之前，推入ISR的位数。 写入值32时实际写入0。	读写	0x00
19	<b>OUT_SHIFTDIR</b> : 1表示输出移位寄存器向右移，0表示向左移。	读写	0x1
18	<b>IN_SHIFTDIR</b> : 1表示输入移位寄存器向右移（数据从左侧进入），0表示向左移。	读写	0x1

位	描述	类型	复位
17	<b>AUTOPULL:</b> 当输出移位寄存器清空时自动拉取，即在执行导致输出移位计数器达到或超过PULL_THRESH的OUT指令时或随后执行。	读写	0x0
16	<b>AUTOPUSH:</b> 当输入移位寄存器填满时自动推送，即在导致输入移位计数器达到或超过PUSH_THRESH的IN指令时触发。	读写	0x0
15	<p><b>FJOIN_RX_PUT:</b> 若设置为1，禁用此状态机的RX FIFO，使其存储可供状态机通过 <code>put</code> 指令进行随机写访问，且除非同时设置FJOIN_RX_GET，处理器可通过RXFx_PUTGETy寄存器进行随机读访问。</p> <p>若FJOIN_RX_PUT和FJOIN_RX_GET均被设置，则RX FIFO的寄存器可被状态机随机读写，但处理器完全无法访问。</p> <p>设置此位将清除FJOIN_TX和FJOIN_RX位。</p>	读写	0x0
14	<p><b>FJOIN_RX_GET:</b> 若设置为1，禁用此状态机的RX FIFO，使其存储可供状态机通过 <code>get</code> 指令进行随机读访问，且除非同时设置FJOIN_RX_PUT，处理器可通过RXFx_PUTGETy寄存器进行随机写访问。</p> <p>若FJOIN_RX_PUT和FJOIN_RX_GET均被设置，则RX FIFO的寄存器可被状态机随机读写，但处理器完全无法访问。</p> <p>设置此位将清除FJOIN_TX和FJOIN_RX位。</p>	读写	0x0
13:5	保留。	-	-
4:0	<p><b>IN_COUNT:</b> 设置由IN PINS、WAIT PIN或MOV x, PINS指令读取时，不被屏蔽且保持原值的引脚数量。IN PINS、WAIT PIN或MOV x, PINS指令。</p> <p>例如，IN_COUNT为5表示IN引脚组的5个最低有效位（位4:0）可见，剩余27个最高有效位则被屏蔽为0。计数为32时，字段值编码为0，故默认行为为不执行任何屏蔽。</p> <p>请注意，此屏蔽是在IN指令通常执行的屏蔽基础上额外应用的。此设置主要用于MOV x, PINS指令，该指令本身不具备引脚屏蔽功能。</p>	读写	0x00

## PIO: SM0\_ADDR, SM1\_ADDR, SM2\_ADDR, SM3\_ADDR 寄存器

偏移量: 0x0d4, 0x0ec, 0x104, 0x11c

表998。SM0\_ADDR,  
SM1\_ADDR,  
SM2\_ADDR,  
SM3\_ADDR寄存器

位	描述	类型	复位
31:5	保留。	-	-
4:0	状态机当前指令地址 N	只读	0x00

## PIO: SM0\_INSTR、SM1\_INSTR、SM2\_INSTR、SM3\_INSTR 寄存器

偏移量: 0x0d8、0x0f0、0x108、0x120

表 999。  
**SM0\_INSTR,**  
**SM1\_INSTR,**  
**SM2\_INSTR,**  
**SM3\_INSTR 寄存器**

位	描述	类型	复位
31:16	保留。	-	-
15:0	读取以查看状态机 N 程序计数器当前指向的指令。  写入以立即执行指令（包括跳转），随后恢复执行。	读写	-

## PIO: SM0\_PINCTRL、SM1\_PINCTRL、SM2\_PINCTRL、SM3\_PINCTRL 寄存器

偏移量: 0x0dc、0x0f4、0x10c、0x124

### 描述

状态机引脚控制

表 1000。  
**SM0\_PINCTRL,**  
**SM1\_PINCTRL,**  
**SM2\_PINCTRL,**  
**SM3\_PINCTRL 寄存器**

位	描述	类型	复位
31:29	<b>SIDESET_COUNT:</b> 延迟/附加设置指令字段中用于附加设置的最高有效位数。包含启用位（如存在）。最少为0（所有延迟位，无侧置），最多为5（所有侧置，无延迟）。	读写	0x0
28:26	<b>SET_COUNT:</b> 由SET断言的引脚数量。范围为0至5（含）。	读写	0x5
25:20	<b>OUT_COUNT:</b> 由OUT PINS、OUT PINDIRS或MOV PINS指令断言的引脚数量。范围为0至32（含）。	读写	0x00
19:15	<b>IN_BASE:</b> 映射到状态机IN数据总线最低有效位的引脚。编号较高的引脚依次映射到更高有效位，且对引脚编号应用模32运算。	读写	0x00
14:10	<b>SIDESET_BASE:</b> 受侧置操作影响的编号最低的引脚。指令侧置/延迟字段的最高有效位（最多5位，由SIDESET_COUNT决定）用于侧置数据，剩余最低有效位用于延迟。侧置部分的最低有效位写入此引脚，更高有效位写入编号更高的引脚。	读写	0x00
9:5	<b>SET_BASE:</b> 将受 SET PINS 或 SET PINDIRS 指令影响的编号最低的引脚。写入此引脚的数据是 SET 数据的最低有效位。	读写	0x00
4:0	<b>OUT_BASE:</b> 将受 OUT PINS、OUT PINDIRS 或 MOV PINS 指令影响的编号最低的引脚。写入此引脚的数据始终是 OUT 或 MOV 数据的最低有效位。	读写	0x00

## PIO: RXF0\_PUTGET0 寄存器

偏移: 0x128

表 1001。  
**RXF0\_PUTGET0 寄存器**

位	描述	类型	复位
31:0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ_OIN_RX_GET时，可直接读写SM0的RX FIFO第0条目	读写	0x00000000

## PIO: RXF0\_PUTGET1 寄存器

偏移: 0x12c

表 1002。  
RXF0\_PUTGET1  
寄存器

位	描述	类型	复位
31:0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET时，可直接读写SM0的RX FIFO第1条目	读写	0x00000000

## PIO: RXF0\_PUTGET2 寄存器

偏移量：0x130

表 1003。  
RXF0\_PUTGET2  
寄存器

位	描述	类型	复位
31:0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET时，可直接读写SM0的RX FIFO第2条目	读写	0x00000000

## PIO: RXF0\_PUTGET3 寄存器

偏移量：0x134

表 1004。  
RXF0\_PUTGET3  
寄存器

位	描述	类型	复位
31:0	当设置SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET时，可直接读写SM0的RX FIFO第3条目	读写	0x00000000

## PIO: RXF1\_PUTGET0 寄存器

偏移：0x138

表 1005。  
RXF1\_PUTGET0  
寄存器

位	描述	类型	复位
31:0	对SM1的RX FIFO条目0的直接读/写访问（条件是 SHIFTCTRL_FJOIN_RX_PUT 异或 SHIFTCTRL_FJOIN_RX_GET 被设置）。	读写	0x00000000

## PIO: RXF1\_PUTGET1 寄存器

偏移：0x13C

表 1006。  
RXF1\_PUTGET1  
寄存器

位	描述	类型	复位
31:0	当设置 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET 时，可直接读写 SM1 的 RX FIFO 条目 1。	读写	0x00000000

## PIO: RXF1\_PUTGET2 寄存器

偏移：0x140

表 1007。  
RXF1\_PUTGET2  
寄存器

位	描述	类型	复位
31:0	当设置 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET 时，可直接读写 SM1 的 RX FIFO 条目 2。	读写	0x00000000

## PIO: RXF1\_PUTGET3 寄存器

偏移：0x144

表1008  
RXF1\_PUTGET3  
寄存器

位	描述	类型	复位
31:0	当设置 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET 时，可直接读写 SM1 的 RX FIFO 条目 3。	读写	0x00000000

## PIO: RXF2\_PUTGET0寄存器

偏移: 0x148

表1009  
RXF2\_PUTGET0  
寄存器

位	描述	类型	复位
31:0	当设置 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET 时，可直接读写 SM2 的 RX FIFO 条目 0。	读写	0x00000000

## PIO: RXF2\_PUTGET1寄存器

偏移: 0x14c

表1010  
RXF2\_PUTGET1  
寄存器

位	描述	类型	复位
31:0	当设置 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET 时，可直接读写 SM2 的 RX FIFO 条目 1。	读写	0x00000000

## PIO: RXF2\_PUTGET2 寄存器

偏移: 0x150

表1011。  
RXF2\_PUTGET2  
寄存器

位	描述	类型	复位
31:0	对 SM2 的 RX FIFO 条目 2 的直接读写访问，前提是 SHIFTCTRL_FJOIN_RX_PUT 异或 SHIFTCTRL_FJOIN_RX_GET 被设置)。	读写	0x00000000

## PIO: RXF2\_PUTGET3 寄存器

偏移: 0x154

表1012。  
RXF2\_PUTGET3  
寄存器

位	描述	类型	复位
31:0	当设置 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJ OIN_RX_GET 时，可直接读写 SM2 的 RX FIFO 条目 3。	读写	0x00000000

## PIO: RXF3\_PUTGET0 寄存器

偏移: 0x158

表1013。  
RXF3\_PUTGET0  
寄存器

位	描述	类型	复位
31:0	当设置了 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_ FJOIN_RX_GET 时，可直接读写 SM3 的 RX FIFO 条目 0。	读写	0x00000000

## PIO: RXF3\_PUTGET1 寄存器

偏移: 0x15c

表 1014。  
RXF3\_PUTGET1  
寄存器

位	描述	类型	复位
31:0	当设置了 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET 时，可直接读写 SM3 的 RX FIFO 条目 1。	读写	0x00000000

## PIO: RXF3\_PUTGET1 寄存器

偏移: 0x160

表 1015。  
RXF3\_PUTGET2  
寄存器

位	描述	类型	复位
31:0	当设置了 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET 时，可直接读写 SM3 的 RX FIFO 条目 2。	读写	0x00000000

## PIO: RXF3\_PUTGET2 寄存器

偏移: 0x164

表 1016。  
RXF3\_PUTGET3  
寄存器

位	描述	类型	复位
31:0	当设置了 SHIFTCTRL_FJOIN_RX_PUT xor SHIFTCTRL_FJOIN_RX_GET 时，可直接读写 SM3 的 RX FIFO 条目 3。	读写	0x00000000

## PIO: GPIOBASE 寄存器

偏移: 0x168

表 1017。  
GPIOBASE 寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	在系统 GPIO 编号中重新定位 GPIO 0（从 PIO 视角），以使 PIO 可访问超过 32 个 GPIO。  仅支持值 0 和 16（仅可写第 4 位）。	读写	0x0
3:0	保留。	-	-

## PIO: INTR 寄存器

偏移: 0x16c

### 描述

原始中断

表 1018. INTR  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	只读	0x0
14	<b>SM6</b>	只读	0x0
13	<b>SM5</b>	只读	0x0
12	<b>SM4</b>	只读	0x0
11	<b>SM3</b>	只读	0x0
10	<b>SM2</b>	只读	0x0
9	<b>SM1</b>	只读	0x0
8	<b>SM0</b>	只读	0x0

位	描述	类型	复位
7	<b>SM3_TXNFULL</b>	只读	0x0
6	<b>SM2_TXNFULL</b>	只读	0x0
5	<b>SM1_TXNFULL</b>	只读	0x0
4	<b>SM0_TXNFULL</b>	只读	0x0
3	<b>SM3_RXNEMPTY</b>	只读	0x0
2	<b>SM2_RXNEMPTY</b>	只读	0x0
1	<b>SM1_RXNEMPTY</b>	只读	0x0
0	<b>SM0_RXNEMPTY</b>	只读	0x0

## PIO: IRQ0\_INTE 寄存器

偏移: 0x170

### 描述

irq0 中断使能

表 1019  
IRQ0\_INTE 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	读写	0x0
14	<b>SM6</b>	读写	0x0
13	<b>SM5</b>	读写	0x0
12	<b>SM4</b>	读写	0x0
11	<b>SM3</b>	读写	0x0
10	<b>SM2</b>	读写	0x0
9	<b>SM1</b>	读写	0x0
8	<b>SM0</b>	读写	0x0
7	<b>SM3_TXNFULL</b>	读写	0x0
6	<b>SM2_TXNFULL</b>	读写	0x0
5	<b>SM1_TXNFULL</b>	读写	0x0
4	<b>SM0_TXNFULL</b>	读写	0x0
3	<b>SM3_RXNEMPTY</b>	读写	0x0
2	<b>SM2_RXNEMPTY</b>	读写	0x0
1	<b>SM1_RXNEMPTY</b>	读写	0x0
0	<b>SM0_RXNEMPTY</b>	读写	0x0

## PIO: IRQ0\_INTF 寄存器

偏移: 0x174

### 描述

irq0 中断强制

表 1020  
IRQ0\_INTF 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	读写	0x0
14	<b>SM6</b>	读写	0x0
13	<b>SM5</b>	读写	0x0
12	<b>SM4</b>	读写	0x0
11	<b>SM3</b>	读写	0x0
10	<b>SM2</b>	读写	0x0
9	<b>SM1</b>	读写	0x0
8	<b>SM0</b>	读写	0x0
7	<b>SM3_RXNEMPTY</b>	读写	0x0
6	<b>SM2_RXNEMPTY</b>	读写	0x0
5	<b>SM1_RXNEMPTY</b>	读写	0x0
4	<b>SM0_RXNEMPTY</b>	读写	0x0
3	<b>SM3_TXNFULL</b>	读写	0x0
2	<b>SM2_TXNFULL</b>	读写	0x0
1	<b>SM1_TXNFULL</b>	读写	0x0
0	<b>SM0_TXNFULL</b>	读写	0x0

## PIO: IRQ0\_INTS 寄存器

偏移: 0x178

### 描述

irq0 掩码及强制后的中断状态

表 1021  
IRQ0\_INTS 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	只读	0x0
14	<b>SM6</b>	只读	0x0
13	<b>SM5</b>	只读	0x0
12	<b>SM4</b>	只读	0x0
11	<b>SM3</b>	只读	0x0
10	<b>SM2</b>	只读	0x0
9	<b>SM1</b>	只读	0x0
8	<b>SM0</b>	只读	0x0
7	<b>SM3_RXNEMPTY</b>	只读	0x0
6	<b>SM2_RXNEMPTY</b>	只读	0x0
5	<b>SM1_RXNEMPTY</b>	只读	0x0
4	<b>SM0_RXNEMPTY</b>	只读	0x0

位	描述	类型	复位
3	<b>SM3_RXNEMPTY</b>	只读	0x0
2	<b>SM2_RXNEMPTY</b>	只读	0x0
1	<b>SM1_RXNEMPTY</b>	只读	0x0
0	<b>SM0_RXNEMPTY</b>	只读	0x0

## PIO：IRQ1\_INTE 寄存器

偏移：0x17c

### 描述

用于 irq1 的中断使能

表 1022  
IRQ1\_INTE 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	读写	0x0
14	<b>SM6</b>	读写	0x0
13	<b>SM5</b>	读写	0x0
12	<b>SM4</b>	读写	0x0
11	<b>SM3</b>	读写	0x0
10	<b>SM2</b>	读写	0x0
9	<b>SM1</b>	读写	0x0
8	<b>SM0</b>	读写	0x0
7	<b>SM3_TXNFULL</b>	读写	0x0
6	<b>SM2_TXNFULL</b>	读写	0x0
5	<b>SM1_TXNFULL</b>	读写	0x0
4	<b>SM0_TXNFULL</b>	读写	0x0
3	<b>SM3_RXNEMPTY</b>	读写	0x0
2	<b>SM2_RXNEMPTY</b>	读写	0x0
1	<b>SM1_RXNEMPTY</b>	读写	0x0
0	<b>SM0_RXNEMPTY</b>	读写	0x0

## PIO：IRQ1\_INTF 寄存器

偏移：0x180

### 描述

用于 irq1 的中断强制

表 1023  
IRQ1\_INTF 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	读写	0x0
14	<b>SM6</b>	读写	0x0
13	<b>SM5</b>	读写	0x0

位	描述	类型	复位
12	<b>SM4</b>	读写	0x0
11	<b>SM3</b>	读写	0x0
10	<b>SM2</b>	读写	0x0
9	<b>SM1</b>	读写	0x0
8	<b>SM0</b>	读写	0x0
7	<b>SM3_TXNFULL</b>	读写	0x0
6	<b>SM2_TXNFULL</b>	读写	0x0
5	<b>SM1_TXNFULL</b>	读写	0x0
4	<b>SM0_TXNFULL</b>	读写	0x0
3	<b>SM3_RXNEMPTY</b>	读写	0x0
2	<b>SM2_RXNEMPTY</b>	读写	0x0
1	<b>SM1_RXNEMPTY</b>	读写	0x0
0	<b>SM0_RXNEMPTY</b>	读写	0x0

## PIO：IRQ1\_INTS 寄存器

偏移：0x184

### 描述

irq1 掩码和强制后的中断状态

表 1024。  
IRQ1\_INTS 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>SM7</b>	只读	0x0
14	<b>SM6</b>	只读	0x0
13	<b>SM5</b>	只读	0x0
12	<b>SM4</b>	只读	0x0
11	<b>SM3</b>	只读	0x0
10	<b>SM2</b>	只读	0x0
9	<b>SM1</b>	只读	0x0
8	<b>SM0</b>	只读	0x0
7	<b>SM3_TXNFULL</b>	只读	0x0
6	<b>SM2_TXNFULL</b>	只读	0x0
5	<b>SM1_TXNFULL</b>	只读	0x0
4	<b>SM0_TXNFULL</b>	只读	0x0
3	<b>SM3_RXNEMPTY</b>	只读	0x0
2	<b>SM2_RXNEMPTY</b>	只读	0x0
1	<b>SM1_RXNEMPTY</b>	只读	0x0
0	<b>SM0_RXNEMPTY</b>	只读	0x0

# 第12章 外设

## 12.1. UART

Arm 文档

摘自 PrimeCell UART (PL011) 技术参考手册，已获授权使用。

RP2350 配备两个基于 Arm PrimeCell UART (PL011) (版本 r1p5) 的相同 UART 外设实例。

每个实例支持以下功能：

- 独立的 32×8 发送 (TX) 和 32×12 接收 (RX) FIFO
- 可编程波特率发生器，时钟源为 `clk_peri` (见图33)
- 标准异步通信位（起始位、停止位、奇偶校验），发送时添加，接收时移除
- 线路中断检测
- 可编程串行接口（5、6、7 或 8 位）
- 1 或 2 个停止位
- 可编程硬件流控

每个 UART 均可连接至多个 GPIO 引脚，详见第9.4节的 GPIO 复用表。GPIO 复用连接使用 UART 实例名称为前缀 `uart0_` 或 `uart1_`，包含以下内容：

- 传输数据 `tx` (在以下章节中称为 `UARTTXD`)
- 接收数据 `rx` (在以下章节中称为 `UARTRXD`)
- 输出流控制 `rts` (在以下章节中称为 `nUARTRTS`)
- 输入流控制 `cts` (在以下章节中称为 `nUARTCTS`)。PL011的调制解调器

模式和IrDA模式不被支持。

`UARTCLK`由`clk_peri`驱动，`PCLK`由系统时钟 `clk_sys`驱动 (参见图33)。

### 12.1.1. 概述

UART执行以下功能：

- 对从外围设备接收的数据进行串行转并行转换。
- 对传输至外围设备的数据进行并行转串行转换。

CPU通过AMBA APB接口读写数据及控制/状态信息。收发路径均配备内部FIFO缓存器，在发送和接收模式中独立存储最多32字节。

UART具备以下功能：

- 包含一个可编程波特率发生器，从UART内部参考时钟输入 `UARTCLK`产生通用的发送和接收内部时钟
- 提供与行业标准16C650 UART设备类似的功能
- 支持UART模式下最大波特率为 `UARTCLK / 16` (在125MHz时为7.8 Mbaud)

UART的操作及波特率数值由行控制寄存器（**UARTLCR\_H**）和波特率除数寄存器：整数波特率寄存器（**UARTIBRD**）及分数波特率寄存器（**UARTFBRD**）控制。

UART可产生：

- 可单独屏蔽的来自接收（包括超时）、发送、调制解调器状态及错误条件的中断
- 单一组合中断，若任何单独中断被触发且未屏蔽，则该输出被断言
- 用于与直接存储器访问（DMA）控制器接口的DMA请求信号

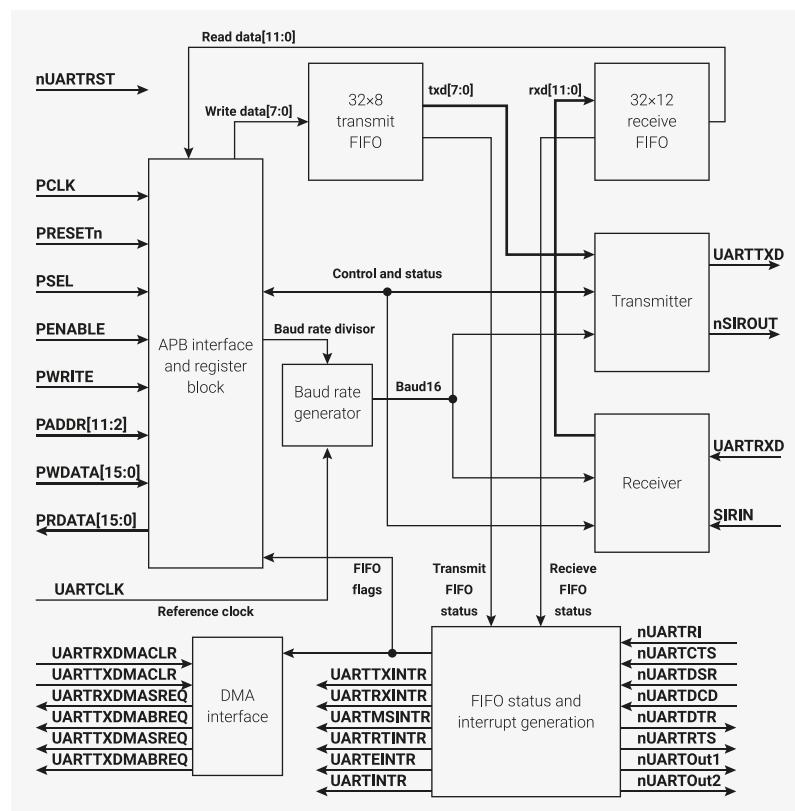
若接收过程中发生帧错误、奇偶校验错误或中断错误，相应错误位将被设置并存储于FIFO中。若发生溢出情况，溢出寄存器位会立即被设置，且会阻止FIFO数据被覆盖。

您可以将FIFO配置为1字节深度，以提供传统的双缓冲UART接口。

具备可编程硬件流控制功能，利用 **nUARTCTS** 输入和 **nUARTRTS** 输出自动控制串行数据流。

## 12.1.2. 功能描述

图 63. UART 方框图  
为清晰起见，未显示测试逻辑



### 12.1.2.1. AMBA APB接口

AMBA APB接口为访问状态/控制寄存器及发送和接收FIFO生成读写译码信号。

### 12.1.2.2. 寄存器块

寄存器块存储通过AMBA APB接口写入或将被读取的数据。

### 12.1.2.3. 波特率发生器

波特率发生器包含自由运行计数器，用以生成内部时钟信号：Baud16与IrLPBaud16。Baud16为UART的发送与接收控制提供时序信息。Baud16是一种脉冲流，脉冲宽度为一个UARTCLK时钟周期，频率为波特率的16倍。

### 12.1.2.4. 发送 FIFO

发送 FIFO 是一个8位宽、深度为32个位置的先进先出存储器缓冲区。通过 APB 接口写入的 CPU 数据存储于 FIFO 中，直至被发送逻辑读取。禁用时，发送 FIFO 表现为一个一字节暂存寄存器。

### 12.1.2.5. 接收 FIFO

接收 FIFO 是一个12位宽、深度为32个位置的先进先出存储器缓冲区。接收逻辑将接收到的数据及其相应错误位存储于接收 FIFO 中，直至 CPU 通过 APB 接口读取。禁用时，接收 FIFO 表现为一个一字节暂存寄存器。

### 12.1.2.6. 发送逻辑

发送逻辑对从发送 FIFO 读取的数据执行并行到串行的转换。控制逻辑按以下顺序输出串行比特流：

1. 起始位
2. 数据位（最低有效位（LSB）优先）
3. 校验位
4. 停止位根据控制寄存器中所编程的配置确定

### 12.1.2.7. 接收逻辑

接收逻辑在检测到有效起始脉冲后，对接收的比特流执行串行转并行转换。接收逻辑包括超限检测、校验检测、帧错误检测及线路中断检测；您可以在随数据写入接收FIFO时附带的状态中查阅这些检测结果。

### 12.1.2.8. 中断生成逻辑

UART向处理器中断控制器生成各个可屏蔽的高电平有效中断信号。为生成组合中断，UART输出各个中断请求的逻辑或（OR）函数。

更多信息详见第12.1.6节。

### 12.1.2.9. DMA接口

UART提供作为UART DMA的接口以连接DMA控制器；更多信息详见第12.1.5节。

### 12.1.2.10. 寄存器与逻辑的同步

UART支持时钟的异步和同步操作，分别为 **PCLK** 与 **UARTCLK**。UART实现了常时同步寄存器及握手逻辑。此设计对性能与面积的影响极小。UART对数据流双向的控制信号进行同步（从 **PCLK** 域至 **UARTCLK** 域，以及从 **UARTCLK** 域至 **PCLK** 域）。

## 12.1.3. 操作

### 12.1.3.1. 时钟信号

所选 **UARTCLK** 频率必须符合所需的波特率范围：

- **FUARTCLK** (最小值)  $\geq 16 \times \text{baud\_rate} (最大值)$
- **FUARTCLK** (最大值)  $\leq 16 \times 65535 \times \text{baud\_rate} (最小值)$

例如，波特率范围为 110 baud 至 460800 baud 时，**UARTCLK** 频率必须介于 7.3728MHz 至 115.34MHz 之间。

为使用所有波特率，**UARTCLK** 频率必须处于规定的误差限度内。

针对 **PCLK** 与 **UARTCLK** 的时钟频率比率亦有限制。**UARTCLK** 的频率不得超过 **PCLK** 频率的 5/3 倍，具体如下：

- $\text{FUARTCLK} \leq 5/3 \times \text{FPCLK}$

例如，在 UART 模式下，当 **UARTCLK** 为 14.7456MHz 时，为生成 921600 波特率，**PCLK** 必须大于或等于 8.85276MHz。此条件确保 UART 有足够时间将接收数据写入接收 FIFO 缓冲区。

### 12.1.3.2 UART 操作

控制数据写入 UART 线路控制寄存器 **UARTLCR**。该寄存器内部宽度为 30 位，但通过 APB 接口对以下寄存器的写访问提供外部接口：

- **UARTLCR\_H**，具体定义如下：
  - 传输参数
  - 字长
  - 缓冲模式
  - 发送停止位数
  - 奇偶校验模式
  - 断帧生成
- **UARTIBRD**，定义整数波特率分频器
- **UARTFBRD**，定义小数波特率分频器

#### 12.1.3.2.1. 小数波特率分频器

波特率分频器是一个由 16 位整数和 6 位小数组成的 22 位数。波特率发生器使用波特率分频器确定比特周期。小数波特率分频器允许使用频率高于 3.6864MHz 的任意时钟作为 **UARTCLK**，同时仍可生成所有标准波特率。

16 位整数写入整数波特率寄存器 **UARTIBRD**。6 位小数部分写入小数波特率寄存器 **UARTFBRD**。波特率分频器与 **UARTCLK** 之间的关系如下：

波特率除数 =  $\text{UARTCLK}/(16 \times \text{波特率}) = BRD_I + BRD_F$  其中  $BRD_I$  为整数部分， $BRD_F$  为如图64所示，用小数点分隔的分数部分。

图 64. 波特率分频器



要计算6位数（），将所需波特率除数的小数部分乘以64（，其中为 **UARTFBRD** 寄存器的位宽），并加0.5<sup>12</sup>考虑舍入误差：

$$m = \text{integer}(BRD_F \times 2^n + 0.5)$$

UART生成一个内部时钟使能信号Baud16。该信号是一串宽度为 **UARTCLK** 的脉冲，平均频率为所需波特率的16倍。将该信号除以16以生成发送时钟。波特率除数较低时比特周期较短，波特率除数较高时比特周期较长。

#### 12.1.3.2.2. 数据传输或接收

UART 使用两个32字节的 FIFO 分别存储接收和发送的数据。接收FIFO的每个字符额外包含四位状态信息。对于发送，数据写入发送FIFO。若UART已启用，将按照线路控制寄存器 **UARTLCR\_H** 中指示的参数启动数据帧传输。数据将持续传输，直至发送FIFO无数据。数据写入发送FIFO后BUSY信号立即置高（即FIFO非空），且在数据传输期间保持高电平。仅当发送FIFO为空且最后一个字符（包括停止位）已从移位寄存器传输完毕时，BUSY信号被清除。尽管UART可能已不再启用，BUSY信号仍可保持高电平。

对于每次数据采样，执行三次读数，并保留多数值。以下段落中定义了中间采样点，并在其两侧各采集一个样本。

当接收器空闲（**UARTRXD**持续为1，处于标记状态）且在数据输入端检测到低电平（接收到起始位）时，在Baud16时钟使能下，接收计数器开始计数。UART模式下于该计数器第八个周期采样数据，SIR模式下于第四个周期采样，以适应更短的逻辑0脉冲（位周期中点）。

若 **UARTRXD** 在Baud16的第八个周期仍为LOW，则起始位有效；否则视为假起始位并予以忽略。

若起始位有效，后续数据位依据数据字符的编程长度，于Baud16的每第16个周期（即一个比特周期之后）采样。若启用奇偶校验模式，则随后校验奇偶校验位。

最后，若 **UARTRXD** 为HIGH，则确认有效停止位；否则发生帧错误。接收到完整字后，数据存入接收FIFO，同时关联该字的任何错误位。

#### 12.1.3.2.3. 错误位

接收FIFO在第8位（帧错误）、第9位（奇偶校验错误）及第10位（中断错误）存储三个错误位，每位对应特定字符。接收FIFO第11位存储附加错误位，表示溢出错误。

#### 12.1.3.2.4. 超限位

超限位与接收FIFO中的字符无关。当FIFO已满且下一个字符完全接收进移位寄存器时，超限错误被设置。移位寄存器中的数据将被覆盖，但不会写入FIFO。当FIFO中出现空闲位置时，另一个字符被接收，超限位的状态与该接收字符一起复制到接收FIFO中。随后超限状态被清除。

表1025列出了接收FIFO的各个位功能。

表1025。接收  
FIFO位功能

FIFO位	功能
11	超限指示
10	断帧错误
9	奇偶校验错误
8	帧错误
7:0	接收数据

#### 12.1.3.2.5. 禁用FIFO

UART收发两端的底层条目均等同于一个1字节的保持寄存器。

您可以操作标志以禁用FIFO，从而将FIFO底层条目作为一个1字节寄存器使用。

然而，这并未物理禁用FIFO。当将FIFO用作1字节寄存器时，写入数据寄存器会绕过保持寄存器，除非发送移位寄存器已被使用。

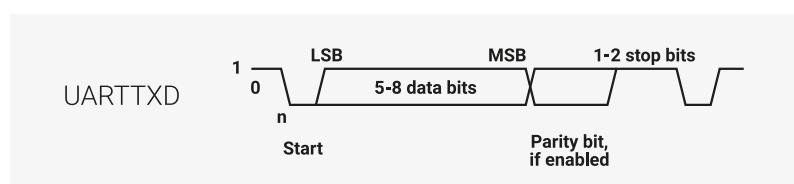
#### 12.1.3.2.6. 系统及诊断环回测试

要执行UART数据的环回测试，请在控制寄存器 [UARTCR](#) 中将环回使能（Loop Back Enable, LBE）位设为1。

通过 [UARTTXD](#) 输出的数据将在 [UARTRXD](#) 输入端接收。

#### 12.1.3.3. UART字符帧

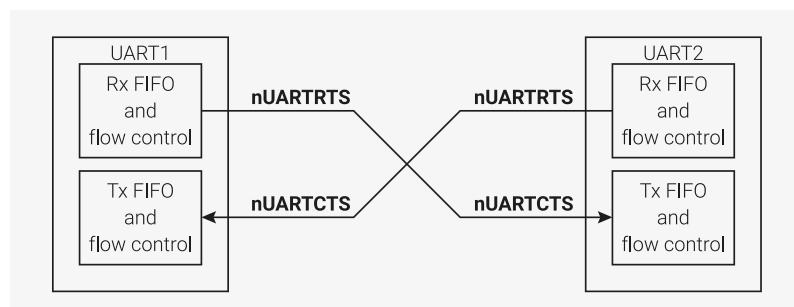
图65. UART  
字符帧。



#### 12.1.4. UART 硬件流控

全选硬件流控功能使您能够通过 [nUARTRTS](#) 输出信号和 [nUARTCTS](#) 输入信号控制串行数据流。图66展示了使用硬件流控在两个设备间进行通信的方式：

图66。两个相似  
设备之间的硬件流  
控示意。



启用RTS流控时，[nUARTRTS](#)将在接收FIFO填充至设定水位线之前被置为激活状态。当启用CTS流控时，发送器仅在[UARTCTS](#)断言时方可发送数据。

硬件流控可通过控制寄存器 [UARTCR](#) 中的 [RTSEn](#) 和 [CTSEn](#) 位进行选择。表 1026 显示了如何配置 [UARTCR](#) 寄存器位以启用 RTS 和/或 CTS。

表 1026。用于启用和禁用硬件流控的控制位。

UARTCR 寄存器位		
CTSEn	RTSEn	描述
1	1	同时启用 RTS 和 CTS 流控
1	0	仅启用 CTS 流控
0	1	仅启用 RTS 流控
0	0	同时禁用 RTS 和 CTS 流控

#### ① 注意

启用 RTS 流控时，软件不可使用控制寄存器（UARTCR）中的 RTSEn 位来控制 UARTRTS 的状态。

#### 12.1.4.1. RTS 流控

RTS 流控逻辑与可编程接收 FIFO 水位线级别相关联。

当 RTS 流控被禁用时，接收 FIFO 将接收数据直至满，或无更多数据传输。

当 RTS 流控启用时，nUARTRTS 信号将在接收 FIFO 填充至水位线前被断言。当接收 FIFO 达到水位线时，nUARTRTS 信号将取消断言。这表示 FIFO 已无更多空间用于接收数据。预计数据传输将在当前字符发送完成后停止。

当接收 FIFO 低于水位线时，nUARTRTS 信号将重新断言。

#### 12.1.4.2. CTS 流控

CTS 流控逻辑与 nUARTCTS 信号相关联。

当 CTS 流控被禁用时，发射器将传输数据直到发送 FIFO 为空。

当 CTS 流控启用时，发射器在发送每个字节前会检查 nUARTCTS 信号。只有在 nUARTCTS 信号被断言时，才传输该字节。只要发送 FIFO 非空且 nUARTCTS 被断言，数据将持续传输。如果发送 FIFO 为空且 nUARTCTS 信号被断言，则不传输数据。如果在传输过程中 nUARTCTS 信号被取消断言，发送器将在完成当前字符传输后停止。

#### 12.1.5. UART DMA 接口

UART 提供与 DMA 控制器连接的接口。UART 的 DMA 操作通过 DMA 控制寄存器 UARTDMACR 进行控制。DMA 接口包括以下信号：

接收：

**UARTRXDMASREQ**

单字符 DMA 传输请求，由 UART 断言。对于接收，一字符最多包含 12 位。

当接收 FIFO 至少包含一个字符时，该信号被断言。

**UARTRXDMABREQ**

突发 DMA 传输请求，由 UART 断言。当接收 FIFO 中的字符数超过设定水位线时，该信号被断言。可以通过中断 FIFO 水平选择寄存器 UARTIFLS 为每个 FIFO 设置水位线。

**UARTRXDMACLR**

DMA请求清除信号，由DMA控制器发送以清除接收请求信号。若请求DMA突发传输，则在突发传输最后一条数据传送时，清除信号被发送。

关于发送：

**UARTTXDMASREQ**

单字符DMA传输请求，由UART发送。对于发送操作，一个字符最多包含八位。当发送FIFO至少有一个空闲位置时，该信号被发送。

**UARTTXDMABREQ**

突发DMA传输请求，由UART发送。当发送FIFO中的字符数量低于水位线时，该信号被发送。可以通过中断FIFO水平选择寄存器**UARTIFLS**为每个FIFO设置水位线。

**UARTTXDMACLR**

DMA请求清除信号，由DMA控制器发送以清除发送请求信号。若请求DMA突发传输，则在突发传输最后一条数据传送时，清除信号被发送。

突发传输请求和单次传输请求信号并不互斥：两者可以同时被发送。当接收FIFO超过水位线时，突发传输请求信号与单次传输请求信号均被发送。当接收FIFO低于水位线水平时，仅断言单次传输请求信号。此功能适用于流中剩余待接收字符数少于一次突发传输的情况。

假设水位线设置为四，但剩余待接收字符数为十九。DMA控制器随后进行四次四字符突发传输和三次单字符传输以完成数据流。

**i 注意**

对于剩余的三个字符，UART无法断言突发请求信号。

每个请求信号持续保持断言状态，直至相关DMACLR信号被断言。请求清除信号解除断言后，依据前述条件，请求信号可重新激活。如果UART被禁用或DMA控制寄存器UARTDMACR中的相关DMA使能位 **TXDMAE** 或 **RXDMAE** 被清除，所有请求信号将被解除断言。

若在UART中禁用FIFO，则其以字符模式运行。字符模式限制FIFO一次只能传输一个字符，因此仅支持DMA单次传输模式。在字符模式下，仅可断言**UARTRXDMASREQ** 和 **UARTTXDMASREQ** 请求信号。有关禁用FIFO的详细信息，请参见行控制寄存器UARTLCR\_H。

当UART处于FIFO启用模式时，数据传输可根据配置的水位线水平和FIFO中的数据量，采用单次或突发传输。表1027列出了传输和接收FIFO中，基于水位线水平的**UARTRXDMAREQ** 和 **UARTTXDMAREQ** 触发点。

表1027。传输  
和接收FIFO的DMA  
触发点

水位线水平	突发长度	
	传输（空位置数量 ）	接收（填充位置数量）
1/8	28	4
1/4	24	8
1/2	16	16
3/4	8	24
7/8	4	28

此外，DMA控制寄存器 **UARTDMACR** 中的 **DMAONERR** 位支持接收错误中断功能，

**UARTEINTR。** 当UART错误中断UARTEINTR被触发时，允许屏蔽DMA接收请求输出，**UARTRXDMASREQ**或**UARTRXDMABREQ**。DMA接收请求输出将保持非激活状态，直到**UARTEINTR**被清除。DMA发送请求输出不受影响。

图67。DMA  
传输波形。

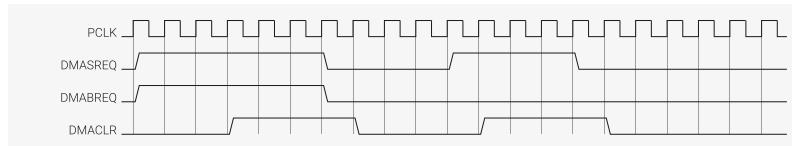


图67显示了单次传输请求和带有相应**DMACLR**信号的突发传输请求的时序图。所有信号均与PCLK同步。为明确起见，假定DMA控制器中请求信号无同步处理。

## 12.1.6. 中断

UART产生十一种可屏蔽中断。在RP2350中，仅连接了合并中断输出，**UARTINTR**。

要启用或禁用单个中断，请修改中断屏蔽设置/清除寄存器 **UARTIMSC**中的掩码位。将相应的掩码位设置为高电平以使能中断。

发送和接收数据流中断 **UARTRXINTR**和 **UARTTXINTR**已从状态中断中分离。

这使您能够使用 **UARTRXINTR**和 **UARTTXINTR**根据FIFO触发级别读取或写入数据。

错误中断 **UARTEINTR**可在接收数据发生错误时触发。可能存在多种错误状态。

调制解调器状态中断 **UARTMSINTR**是所有单个调制解调器状态信号的组合中断。

单个中断源的状态可以从原始中断状态寄存器 **UARTRIS**或屏蔽中断状态寄存器 **UARTMIS**读取。

### 12.1.6.1. **UARTMSINTR**

当任何调制解调器状态信号（**nUARTCTS**、**nUARTDCD**、**nUARTDSR**和 **nUARTRI**）发生变化时，调制解调器状态中断将被断言。要清除调制解调器状态中断，请向中断清除寄存器（**UARTICR**）中对应于产生中断的调制解调器状态信号的位写入1。

### 12.1.6.2. **UARTRXINTR**

当发生以下任一情况时，接收中断状态将发生变化：

- FIFO已启用且接收FIFO达到设定的触发级别。这将使接收中断变为高电平。要清除接收中断，请从接收FIFO读取数据，直到其低于触发级别。
- FIFO已禁用（深度仅为一个位置）且接收到数据，从而填满接收FIFO，这将使接收中断变为高电平。要清除接收中断，请执行一次从接收FIFO的读取操作。

以上两种情况均可通过手动操作清除中断。

### 12.1.6.3. **UARTTXINTR**

当发生以下任一情况时，发送中断状态将发生变化：

- FIFO已启用且发送FIFO的内容等于或低于设定的触发级别。此操作将使发送中断置为高电平。要清除发送中断，请向发送FIFO写入数据，直到其超过

触发电平。

- FIFO被禁用（深度为单个位置），且发送FIFO中无数据。此操作将发送中断置为高电平。要清除发送中断，请对发送FIFO执行一次写入操作。

以上两种情况均可通过手动操作清除中断。

更新发送FIFO时，可在启用UART和中断之前或之后写入数据。

#### 注意

发送中断基于电平的变化触发，而非电平本身。当中断及UART在未向发送FIFO写入任何数据前即被启用时，中断不会触发。只有在写入数据离开发送FIFO的单一位置且FIFO变为空时，中断才会触发。

#### 12.1.6.4. **UARTRTINTR**

当接收FIFO非空且在32位周期内无新数据接收时，接收超时中断被触发。

接收超时中断在以下情况下被清除：

- 通过读取所有数据或读取保持寄存器，使 FIFO 变为空
- 在中断清除寄存器 **UARTICR** 的相应位写入 1

#### 12.1.6.5. **UARTEINTR**

当 UART 接收数据发生错误时，错误中断将被触发。该中断可能由多种不同的错误条件引起：

- 帧错误
- 奇偶校验错误
- 中断符错误
- 溢出错误

要确定中断原因，请读取原始中断状态寄存器（**UARTRIS**）或屏蔽中断状态寄存器（**UARTMIS**）。要清除中断，请向中断清除寄存器 **UARTICR** 的相关位写入数据（第7至10位为错误清除位）。

#### 12.1.6.6. **UARTINTR**

这些中断还被合并为单一输出，即各个屏蔽中断源的逻辑或功能结果。您可以将此输出连接到系统中断控制器，以便基于单个外设提供另一层屏蔽。

只要任一单独中断被断言且已启用，组合 UART 中断即被断言。

### 12.1.7. 程序员模型

SDK 提供了 **uart\_init** 函数，用于配置特定波特率的 UART。UART 初始化后，用户必须将某个 GPIO 引脚配置为 **UART\_TX** 和 **UART\_RX**。有关 GPIO 功能选择的详细信息，请参见第 9.10.1 节。

初始化 UART 时，**uart\_init** 函数执行以下步骤：

1. 取消复位断言

2. 启用 `clk_peri`
3. 在控制寄存器中设置使能位
4. 启用 FIFO
5. 设置波特率分频值
6. 设置格式

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_uart/uart.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_uart/uart.c) 第 42 至 92 行

```

42 uint uart_init(uart_inst_t *uart, uint baudrate) {
43 invalid_params_if(HARDWARE_UART, uart != uart0 && uart != uart1);
44
45 if (uart_clock_get_hz(uart) == 0) {
46 return 0;
47 }
48
49 uart_reset(uart);
50 uart_unreset(uart);
51
52 uart_set_translate_crlf(uart, PICO_UART_DEFAULT_CRLF);
53
54 // 任何对 LCR 寄存器的写入必须在启用 UART 之前完成
55 uint baud = uart_set_baudrate(uart, baudrate);
56
57 // 将 uart_set_format() 内联处理，因为我们不需要 CR 禁用后再启用
58 // 的保护，且许多用户可能永远不会再次调用此函数，
59 // 通用函数无实际用处，且比这段仅包含少数字节的内联代码体积更大。
60 // code which is only a handful of instructions.
61 //
62 // UART_UARTLCR_H_FEN_BITS 设置已合并，因为它们位于同一寄存器中
63 #ifdef 0
64 uart_set_format(uart, 8, 1, UART_PARITY_NONE);
65 // 启用 FIFO (必须在设置 UARTEN 之前，因为此操作涉及 LCR 寄存器访问)
66 hw_set_bits(&uart_get_hw(uart)->lcr_h, UART_UARTLCR_H_FEN_BITS);
67 #else
68 uint data_bits = 8;
69 uint stop_bits = 1;
70 uint parity = UART_PARITY_NONE;
71 hw_write_masked(&uart_get_hw(uart)->lcr_h,
72 ((data_bits - 5u) << UART_UARTLCR_H_WLEN_LSB) |
73 ((stop_bits - 1u) << UART_UARTLCR_H_STP2_LSB) |
74 (bool_to_bit(parity != UART_PARITY_NONE) << UART_UARTLCR_H_PEN_LSB) |
75 (bool_to_bit(parity == UART_PARITY_EVEN) << UART_UARTLCR_H_EPS_LSB) |
76 UART_UARTLCR_H_FEN_BITS,
77 UART_UARTLCR_H_WLEN_BITS | UART_UARTLCR_H_STP2_BITS |
78 UART_UARTLCR_H_PEN_BITS | UART_UARTLCR_H_EPS_BITS |
79 UART_UARTLCR_H_FEN_BITS);
80 #endif
81
82 // 启用 UART，传输和接收均已启用
83 uart_get_hw(uart)->cr = UART_UARTCR_UARTEN_BITS | UART_UARTCR_TXE_BITS |
84 UART_UARTCR_RXE_BITS;
85 // 始终启用 DREQ 信号——即使 DMA 未监听，也无影响
86 uart_get_hw(uart)->dmacr = UART_UARTDMACR_TXDMAE_BITS | UART_UARTDMACR_RXDMAE_BITS;
87
88 return baud;
89 }
```

### 12.1.7.1. 波特率计算

UART 波特率通过除以 `clk_peri` 计算得出。

如果所需波特率为 115200，且  $\text{UARTCLK} = 125\text{MHz}$ ，则：

$$\text{波特率除数} = (125 \times 10^6) / (16 \times 115200) \approx 67.817$$

因此， $\text{BRDI} = 67$ ， $\text{BRDF} = 0.817$ ，

因此，分数部分  $m = \text{integer}(0.817 \times 64) + 0.5 = 52$

生成的波特率除频器 =  $67 + 52/64 = 67.8125$

$$\text{生成的波特率} = (125 \times 10^6) / (16 \times 67.8125) \approx 115207$$

$$\text{误差} = (\text{abs}(115200 - 115207) / 115200) \times 100 \approx 0.006\%$$

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_uart/uart.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_uart/uart.c) 第155至180行

```

155 uint uart_set_baudrate(uart_inst_t *uart, uint baudrate) {
156 invalid_params_if(HARDWARE_UART, baudrate == 0);
157 uint32_t baud_rate_div = (8 * uart_clock_get_hz(uart) / baudrate) + 1;
158 uint32_t baud_ibrd = baud_rate_div >> 7;
159 uint32_t baud_fbrd;
160
161 if (baud_ibrd == 0) {
162 baud_ibrd = 1;
163 baud_fbrd = 0;
164 } else if (baud_ibrd >= 65535) {
165 baud_ibrd = 65535;
166 baud_fbrd = 0;
167 } else {
168 baud_fbrd = (baud_rate_div & 0x7f) >> 1;
169 }
170
171 uart_get_hw(uart)->ibrd = baud_ibrd;
172 uart_get_hw(uart)->fbrd = baud_fbrd;
173
174 // PL011 需要进行（伪）LCR_H 写操作以锁存除数。
175 // 此处不应实际更改 LCR_H 内容。
176 uart_write_lcr_bits_masked(uart, 0, 0);
177
178 // 详见数据手册
179 返回 (4 * uart_clock_get_hz(uart)) / (64 * baud_ibrd + baud_fbrd);
180 }
```

### 12.1.8. 寄存器列表

UART0 和 UART1 寄存器的基地址分别为 `0x40070000` 和 `0x40078000`（在 SDK 中定义为 `UART0_BASE` 和 `UART1_BASE`）。

表 1028.  
UART 寄存器列表

偏移量	名称	说明
0x000	UARTDR	数据寄存器 UARTDR
0x004	UARTRSR	接收状态寄存器／错误清除寄存器 UARTRSR/UARTECR
0x018	UARTFR	标志寄存器 UARTFR
0x020	UARTILPR	IrDA 低功耗计数器寄存器 UARTILPR

偏移量	名称	说明
0x024	UARTIBRD	整数波特率寄存器 UARTIBRD
0x028	UARTFBRD	分数波特率寄存器 UARTFBRD
0x02c	UARTLCR_H	行控制寄存器，UARTLCR_H
0x030	UARTCR	控制寄存器，UARTCR
0x034	UARTIFLS	中断FIFO级别选择寄存器，UARTIFLS
0x038	UARTIMSC	中断掩码设置/清除寄存器，UARTIMSC
0x03c	UARTRIS	原始中断状态寄存器，UARTRIS
0x040	UARTMIS	掩码中断状态寄存器，UARTMIS
0x044	UARTICR	中断清除寄存器，UARTICR
0x048	UARTDMACR	DMA控制寄存器，UARTDMACR
0xfe0	UARTPERIPHID0	UART外设ID0寄存器
0xfe4	UARTPERIPHID1	UARTPeriphID1 寄存器
0xfe8	UARTPERIPHID2	UARTPeriphID2 寄存器
0xfec	UARTPERIPHID3	UARTPeriphID3 寄存器
0xff0	UARTPCCELLID0	UARTPCCellID0 寄存器
0xff4	UARTPCCELLID1	UARTPCCellID1 寄存器
0xff8	UARTPCCELLID2	UARTPCCellID2 寄存器
0ffc	UARTPCCELLID3	UARTPCCellID3 寄存器

## UART：UARTDR 寄存器

偏移：0x000

### 描述

数据寄存器 UARTDR

表 1029. UARTDR 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>OE</b> : 溢出错误。当接收到数据且接收 FIFO 已满时，此位被置为 1。一旦 FIFO 中出现空位并可写入新字符，该位即被清零为 0。	只读	-
10	<b>BE</b> : 断路错误。当检测到断路条件时，此位被置为 1，表示接收数据输入保持低电平的时间超过完整字传输时间（包括起始位、数据位、奇偶校验位和停止位）。在 FIFO 模式下，此错误关联于 FIFO 顶部的字符。发生断路时，仅有一个 0 字符加载入 FIFO。只有接收数据输入恢复为 1（标记状态）且接收到下一个有效起始位后，下一字符才被允许进入 FIFO。	只读	-
9	<b>PE</b> : 奇偶校验错误。当该位置为1时，表示接收的数据字符的奇偶校验与线路控制寄存器UARTLCR_H中的EPS和SPS位指定的奇偶校验不符。在FIFO模式下，该错误关联于FIFO顶部的字符。	只读	-

位	描述	类型	复位
8	<b>FE</b> : 帧错误。当该位置为1时，表示接收的字符没有有效的停止位（有效停止位为1）。在FIFO模式下，该错误关联于FIFO顶部的字符。	只读	-
7:0	<b>DATA</b> : 接收（读取）数据字符。发送（写入）数据字符。	RWF	-

## UART: UARTRSR寄存器

偏移: 0x004

### 说明

接收状态寄存器/错误清除寄存器，UARTRSR/UARTECR

表1030. UARTRSR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>OE</b> : 溢出错误。如果接收到数据且FIFO已满，该位将被置为1。该位通过写入UARTECR被清零。FIFO内容保持有效，因为当FIFO满时不会写入更多数据，只有移位寄存器的内容被覆盖。CPU现在必须读取数据，以清空FIFO。	WC	0x0
2	<b>BE</b> : 断路错误。如果检测到中断条件，该位被置为1，表示接收的数据输入保持低电平时间超过完整字传输时间（定义为起始位、数据位、奇偶校验位及停止位）。该位在写入UARTECR后被清零。在FIFO模式下，该错误与FIFO顶部字符相关。发生中断时，FIFO中仅加载一个值为0的字符。只有当接收数据输入回到1（标记状态）且接收到下一个有效起始位后，下一字符才被启用。	WC	0x0
1	<b>PE</b> : 奇偶校验错误。当该位置为1时，表示接收数据字符的奇偶校验与UARTECR_H线控制寄存器中EPS和SPS位定义的奇偶校验不匹配。该位通过写入UARTECR被清零。在FIFO模式下，该错误关联于FIFO顶部的字符。	WC	0x0
0	<b>FE</b> : 帧错误。当该位置为1时，表示接收的字符没有有效的停止位（有效停止位为1）。该位通过写入UARTECR被清零。在FIFO模式下，该错误与FIFO顶部的字符相关。	WC	0x0

## UART: UARTFR寄存器

偏移量: 0x018

### 描述

标志寄存器UARTFR

表1031. UARTFR  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>RI</b> : 环指示信号。该位是UART环指示信号nUARTRI（调制解调器状态输入）的反码。即当nUARTRI为低电平时，该位为1。	只读	-

位	描述	类型	复位
7	<b>TXFE</b> : 发送FIFO为空。该位的含义取决于线控制寄存器UARTLCR_H中FEN位的状态。如果FIFO被禁用，当发送保持寄存器为空时，该位被置位。如果FIFO启用，则当发送FIFO为空时，TXFE位被置位。该位不指示发送移位寄存器中是否存在数据。	只读	0x1
6	<b>RXFF</b> : 接收FIFO已满。该位的含义取决于UARTLCR_H寄存器中FEN位的状态。如果FIFO被禁用，当接收保持寄存器满时，该位被置位。如果启用了FIFO，接收FIFO满时，RXFF位将被置位。	只读	0x0
5	<b>TXFF</b> : 发送FIFO已满。此位的含义取决于UARTLCR_H寄存器中FEN位的状态。若FIFO被禁用，则发送保持寄存器满时，该位被置位。如果启用了FIFO，发送FIFO满时，TXFF位将被置位。	只读	0x0
4	<b>RXFE</b> : 接收FIFO为空。此位的含义取决于UARTLCR_H寄存器中FEN位的状态。若FIFO被禁用，则接收保持寄存器为空时，该位被置位。如果启用了FIFO，接收FIFO为空时，RXFE位将被置位。	只读	0x1
3	<b>BUSY</b> : UART忙碌。若该位被置为1，表示UART正忙于传输数据。该位将保持置位，直至包括所有停止位在内的完整字节已从移位寄存器发送完毕。该位在发送FIFO变为非空时即被置位，无论UART是否启用。	只读	0x0
2	<b>DCD</b> : 数据载波检测。该位是UART数据载波检测信号nUARTDCD（调制解调器状态输入）的反码。即当nUARTDCD为低电平时，该位为1。	只读	-
1	<b>DSR</b> : 数据集准备。该位是UART数据集准备信号nUARTDSR（调制解调器状态输入）的反码。即当nUARTDSR为低电平时，该位为1。	只读	-
0	<b>CTS</b> : 清除发送。该位是UART清除发送信号nUARTCTS（调制解调器状态输入）的反码。即当nUARTCTS为低电平时，该位为1。	只读	-

## UART: UARTILPR寄存器

偏移量: 0x020

### 描述

IrDA 低功耗计数器寄存器 UARTILPR

表1032. UARTILPR  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>ILPDVSR</b> : 8位低功耗除数值。这些位在复位时被清零。	读写	0x00

## UART: UARTIBRD 寄存器

偏移量: 0x024

### 说明

整数波特率寄存器 UARTIBRD

表 1033. UARTIBRD  
寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15:0	<b>BAUD_DIVINT</b> : 整数波特率除数。复位时这些位被清零。	读写	0x0000

## UART: UARTFBRD 寄存器

偏移量: 0x028

### 说明

分数波特率寄存器 UARTFBRD

表 1034。  
UARTFBRD 寄存器

位	描述	类型	复位
31:6	保留。	-	-
5:0	<b>BAUD_DIVFRAC</b> : 小数波特率除数。复位时这些位被清零。	读写	0x00

## UART: UARTLCR\_H 寄存器

偏移: 0x02c

### 描述

行控制寄存器，UARTLCR\_H

表 1035。  
UARTLCR\_H 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>SPS</b> : 固定奇偶校验选择。0 = 禁用固定奇偶校验；1 = 如下之一：* 若 EPS 位为 0，则校验位作为 1 发送并校验；* 若 EPS 位为 1，则校验位作为 0 发送并校验。当 PEN 位禁用奇偶校验生成和校验时，该位无效。	读写	0x0
6:5	<b>WLEN</b> : 字长。这些位指示帧中传输或接收的数据位数，具体如下：b11 = 8 位；b10 = 7 位；b01 = 6 位；b00 = 5 位。	读写	0x0
4	<b>FEN</b> : 启用FIFO: 0 = 禁用FIFO（字符模式），即FIFO变为1字节深的保持寄存器；1 = 启用发送和接收FIFO缓冲区（FIFO模式）。	读写	0x0
3	<b>STP2</b> : 选择两个停止位。如果该位设置为1，则在帧结束时发送两个停止位。接收逻辑不会检查是否接收到两个停止位。	读写	0x0
2	<b>EPS</b> : 奇偶校验选择。控制UART在传输和接收期间使用的校验类型：0 = 奇校验。UART生成或检查数据及校验位中‘1’的奇数个数。1 = 偶校验。UART生成或检查数据及校验位中‘1’的偶数个数。当 PEN 位禁用奇偶校验生成和校验时，该位无效。	读写	0x0
1	<b>PEN</b> : 校验使能：0 = 禁用校验且不向数据帧添加校验位；1 = 启用校验生成和检测。	读写	0x0
0	<b>BRK</b> : 发送中断信号。如果该位被置为1，则在完成当前字符传输后，UARTT XD输出端将持续输出低电平。为确保中断命令正确执行，软件必须至少将该位设置两个完整帧周期。正常使用时，该位必须清零（设置为0）。	读写	0x0

## UART: UARTCR 寄存器

偏移量: 0x030

**描述**

控制寄存器, UARTCR

表 1036. UARTCR  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>CTSEN</b> : 使能CTS硬件流控。若该位被置为1，则启用CTS硬件流控。仅当nUARTCTS信号被断言时，数据才会被传输。	读写	0x0
14	<b>RTSEN</b> : 使能RTS硬件流控。若该位被置为1，则启用RTS硬件流控。仅当接收FIFO有可用空间时，才请求数据。	读写	0x0
13	<b>OUT2</b> : 此位为UART Out2 (nUARTOut2) 调制解调器状态信号的反码。即当该位被编程为1时，输出为0。对于DTE，此位可用作振铃指示 (RI)。	读写	0x0
12	<b>OUT1</b> : 该位为UART Out1 (nUARTOut1) 调制解调器状态输出的反码。即当该位被编程为1时，输出为0。对于DTE，此位可用作载波检测 (DCD)。	读写	0x0
11	<b>RTS</b> : 请求发送。该位为UART请求发送信号nUARTRTS调制解调器状态输出的反码。即当该位被编程为1时，nUARTRTS为低电平。	读写	0x0
10	<b>DTR</b> : 数据传输准备就绪。该位为UART数据传输准备信号nUARTDTR调制解调器状态输出的反码。即当该位被编程为1时，nUARTDTR为低电平。	读写	0x0
9	<b>RXE</b> : 接收使能。若此位被置为1，则启用UART的接收部分。数据接收针对UART信号或SIR信号，取决于SIREN位的设置。UART在接收过程中若被禁用，将完成当前字符后再停止。	读写	0x1
8	<b>TXE</b> : 发送使能。若此位被置为1，则启用UART的发送部分。数据传输针对UART信号或SIR信号，取决于SIREN位的设置。UART在传输过程中若被禁用，将完成当前字符后再停止。	读写	0x1
7	<b>LBE</b> : 回环使能。若此位、SIREN位及测试控制寄存器UARTTCR中的SIRTEST位均置为1，则nSIROUT路径将反转后传递至SIRIN路径。须将测试寄存器中的SIRTEST位置为1，方能覆盖标准的半双工SIR操作。此项为正常操作期间访问测试寄存器的必要条件，且在回环测试完成后，SIRTEST必须清零。该功能减少了系统测试过程中所需的外部耦合数量。若该位设为1且SIRTEST位为0，则UARTTxD路径将直接传输至UARTRxD路径。无论处于SIR模式还是UART模式，当该位被设置时，调制解调器的输出也将传输至其输入端。该位在复位时自动清零，以禁用回环功能。	读写	0x0
6:3	保留。	-	-

位	描述	类型	复位
2	<b>SIRLP</b> : SIR低功耗IrDA模式。该位用于选择IrDA编码模式。若该位清零，低电平比特将作为宽度为比特周期3/16的高电平脉冲进行传输。若该位置1，低电平比特将以IrLPBaud16输入信号周期三倍的脉冲宽度传输，无论所选比特率为何。设置该位可降低功耗，但可能会减少传输距离。	读写	0x0
1	<b>SIREN</b> : SIR使能：0 = 禁用IrDA SIR ENDEC。nSIROUT保持低电平（不产生光脉冲），且SIRIN上的信号跳变无效。1 = 启用IrDA SIR ENDEC。数据通过nSIROUT和SIRIN传输和接收。UARTTXD保持高电平，处于标记状态。UARTRXD或调制解调器状态输入上的信号跳变无效。若UARTEN位禁用UART，则该位无效。	读写	0x0
0	<b>UARTEN</b> : UART使能：0 = 禁用UART。如果在传输或接收过程中禁用UART，将完成当前字符后停止。1 = 启用UART。数据传输与接收根据SIREN位的设置，通过UART信号或SIR信号进行。	读写	0x0

## UART: UARTIFLS 寄存器

偏移: 0x034

### 说明

中断FIFO级别选择寄存器，UARTIFLS

表 1037. UARTIFLS 寄存器

位	描述	类型	复位
31:6	保留。	-	-
5:3	<b>RXIFLSEL</b> : 接收中断FIFO级别选择。接收中断的触发点如下：b000 = 接收FIFO达到或超过1/8满；b001 = 接收FIFO达到或超过1/4满；b010 = 接收FIFO达到或超过1/2满；b011 = 接收FIFO达到或超过3/4满；b100 = 接收FIFO达到或超过7/8满；b101-b111 = 保留。	读写	0x2
2:0	<b>TXIFLSEL</b> : 发送中断FIFO级别选择。发送中断的触发点如下：b000 = 发送FIFO达到或低于1/8满；b001 = 发送FIFO达到或低于1/4满；b010 = 发送FIFO达到或低于1/2满；b011 = 发送FIFO达到或低于3/4满；b100 = 发送FIFO达到或低于7/8满；b101-b111 = 保留。	读写	0x2

## UART: UARTIMSC 寄存器

偏移: 0x038

### 说明

中断掩码设置/清除寄存器，UARTIMSC

表 1038. UARTIMSC 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>OEIM</b> : 溢出错误中断屏蔽读取返回UARTOEINTR中断的当前屏蔽状态。写入1时，设置UARTOEINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0

位	描述	类型	复位
9	<b>BEIM:</b> 断点错误中断屏蔽读取返回UARTBEINTR中断的当前屏蔽状态。写入1时，设置UARTBEINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0
8	<b>PEIM:</b> 奇偶校验错误中断屏蔽读取返回UARTPEINTR中断的当前屏蔽状态。写入1时，设置UARTPEINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0
7	<b>FEIM:</b> 帧错误中断屏蔽读取返回UARTFEINTR中断的当前屏蔽状态。写入1时，设置UARTFEINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0
6	<b>RTIM:</b> 接收超时中断掩码。读取返回UARTRTINTR中断的当前掩码。写入1时，设置UARTRTINTR中断掩码；写入0时，清除该掩码。	读写	0x0
5	<b>TXIM:</b> 发送中断掩码。读取返回UARTTXINTR中断的当前掩码。写入1时，设置UARTTXINTR中断掩码；写入0时，清除该掩码。	读写	0x0
4	<b>RXIM:</b> 接收中断掩码。读取返回UARTRXINTR中断的当前掩码。写入1时，设置UARTRXINTR中断掩码；写入0时，清除该掩码。	读写	0x0
3	<b>DSRMIM:</b> nUARTDSR调制解调器中断掩码。读取返回UARTDSRINTR中断的当前掩码。写入1时，设置UARTDSRINTR中断掩码；写入0时，清除该掩码。	读写	0x0
2	<b>DCDMIM:</b> nUARTDCD调制解调器中断屏蔽位。读取操作返回UARTDCDINTR中断的当前屏蔽状态。写入1时，设置UARTDCDINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0
1	<b>CTSMIM:</b> nUARTCTS调制解调器中断屏蔽位。读取操作返回UARTCTSINTR中断的当前屏蔽状态。写入1时，设置UARTCTSINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0
0	<b>RIMIM:</b> nUARTRI调制解调器中断屏蔽位。读取操作返回UARTRIINTR中断的当前屏蔽状态。写入1时，设置UARTRIINTR中断屏蔽；写入0时，清除该屏蔽。	读写	0x0

## UART: UARTRIS寄存器

偏移量: 0x03c

### 描述

原始中断状态寄存器, UARTRIS

表 1039. UARTRIS  
寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>OERIS:</b> 溢出错误中断状态位。返回UARTOEINTR中断的原始中断状态。	只读	0x0
9	<b>BERIS:</b> 断线错误中断状态。返回UARTBEINTR中断的原始中断状态。	只读	0x0
8	<b>PERIS:</b> 奇偶校验错误中断状态。返回UARTPEINTR中断的原始中断状态。	只读	0x0

位	描述	类型	复位
7	<b>FERIS</b> : 帧错误中断状态。返回UARTFEINTR中断的原始中断状态。	只读	0x0
6	<b>RTRIS</b> : 接收超时中断状态。返回UARTRTINTR中断的原始中断状态。	只读	0x0
5	<b>TXRIS</b> : 发送中断状态。返回UARTTXINTR中断的原始中断状态。	只读	0x0
4	<b>RXRIS</b> : 接收中断状态。返回UARTRXINTR中断的原始中断状态。	只读	0x0
3	<b>DSRMMIS</b> : nUARTDSR调制解调器中断状态。返回UARTDSRINTR中断的原始中断状态。	只读	-
2	<b>DCDRMIS</b> : nUARTDCD调制解调器中断状态。返回UARTDCDINTR中断的原始中断状态。	只读	-
1	<b>CTSRMIS</b> : nUARTCTS 调制解调器中断状态。返回 UARTCTSINTR 中断的原始中断状态。	只读	-
0	<b>RIRMIS</b> : nUARTRI 调制解调器中断状态。返回 UARTRIINTR 中断的原始中断状态。	只读	-

## UART: UARMTMIS 寄存器

偏移量: 0x040

### 描述

掩码中断状态寄存器, UARMTMIS

表 1040. UARMTMIS 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>OEMIS</b> : 溢出错误屏蔽中断状态。返回 UARTOEINTR 中断的屏蔽中断状态。	只读	0x0
9	<b>BEMIS</b> : 断开错误屏蔽中断状态。返回 UARTBEINTR 中断的屏蔽中断状态。	只读	0x0
8	<b>PEMIS</b> : 奇偶校验错误屏蔽中断状态。返回 UARTPEINTR 中断的屏蔽中断状态。	只读	0x0
7	<b>FEMIS</b> : 帧错误屏蔽中断状态。返回 UARTFEINTR 中断的屏蔽中断状态。	只读	0x0
6	<b>RTMIS</b> : 接收超时屏蔽中断状态。返回 UARTRTINTR 中断的屏蔽中断状态。	只读	0x0
5	<b>TXMIS</b> : 发送屏蔽中断状态。返回UARTTXINTR中断的屏蔽中断状态。	只读	0x0
4	<b>RXMIS</b> : 接收屏蔽中断状态。返回UARTRXINTR中断的屏蔽中断状态。	只读	0x0
3	<b>DSRMMIS</b> : nUARTDSR调制解调器屏蔽中断状态。返回UARTDSRINTR中断的屏蔽中断状态。	只读	-
2	<b>DCDRMMIS</b> : nUARTDCD调制解调器屏蔽中断状态。返回UARTDCDINTR中断的屏蔽中断状态。	只读	-

位	描述	类型	复位
1	<b>CTSMMS</b> : nUARTCTS调制解调器屏蔽中断状态。返回UARTCTSINTR中断的屏蔽中断状态。	只读	-
0	<b>RIMMIS</b> : nUARTRI调制解调器屏蔽中断状态。返回UARTRIINTR中断的屏蔽中断状态。	只读	-

## UART: UARTICR 寄存器

偏移量: 0x044

### 说明

中断清除寄存器，UARTICR

表1041。UARTICR 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>OEIC</b> : 超限错误中断清除。清除UARTOEINTR中断。	WC	-
9	<b>BEIC</b> : 断点错误中断清除。清除UARTBEINTR中断。	WC	-
8	<b>PEIC</b> : 奇偶校验错误中断清除。清除UARTPEINTR中断。	WC	-
7	<b>FEIC</b> : 帧错误中断清除。清除UARTFEINTR中断。	WC	-
6	<b>RTIC</b> : 接收超时中断清除。清除UARTRTINTR中断。	WC	-
5	<b>TXIC</b> : 发送中断清除。清除UARTTXINTR中断。	WC	-
4	<b>RXIC</b> : 接收中断清除。清除UARTRXINTR中断。	WC	-
3	<b>DSRMIC</b> : nUARTDSR调制解调器中断清除。清除UARTDSRINTR中断 ◦	WC	-
2	<b>DCDMIC</b> : nUARTDCD调制解调器中断清除。清除UARTDCDINTR中断 ◦	WC	-
1	<b>CTSMIC</b> : nUARTCTS调制解调器中断清除。清除UARTCTSINTR中断 ◦	WC	-
0	<b>RIMIC</b> : nUARTRI调制解调器中断清除。清除UARTRIINTR中断。	WC	-

## UART: UARTDMACR 寄存器

偏移量: 0x048

### 说明

DMA控制寄存器，UARTDMACR

表 1042  
UARTDMACR 寄存器

位	描述	类型	复位
31:3	保留。	-	-
2	<b>DMAONERR</b> : 错误时启用的 DMA。若该位设置为 1，当 UART 错误中断触发时，DMA 接收请求输出 UARTRXDMASREQ 或 UARTRXDMABREQ 将被禁用。	读写	0x0
1	<b>TXDMAE</b> : 传输 DMA 使能。若该位设置为 1，传输 FIFO 的 DMA 功能将被启用。	读写	0x0
0	<b>RXDMAE</b> : 接收 DMA 使能。若该位设置为 1，接收 FIFO 的 DMA 功能将被启用。	读写	0x0

## UART：UARTPERIPHID0 寄存器

偏移：0xfe0

### 描述

UART外设ID0寄存器

表 1043  
UARTPERIPHID0  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>PARTNUMBER0</b> : 该位读回值为 0x11	只读	0x11

## UART：UARTPERIPHID1 寄存器

偏移：0xfe4

### 描述

UARTPeriphID1 寄存器

表 1044  
UARTPERIPHID1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:4	<b>DESIGNERO</b> : 这些位读取值为 0x1	只读	0x1
3:0	<b>PARTNUMBER1</b> : 这些位读取值为 0x0	只读	0x0

## UART：UARTPERIPHID2 寄存器

偏移量：0xfe8

### 描述

UARTPeriphID2 寄存器

表 1045。  
UARTPERIPHID2  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:4	版本号：该字段依赖于 UART 版本：r1p0 0x0, r1p1 0x1 r1p3 0x2, r1p4 0x2, r1p5 0x3	只读	0x3
3:0	<b>DESIGNER1</b> : 这些位读取值为 0x4	只读	0x4

## UART：UARTPERIPHID3 寄存器

偏移量：0fec

### 描述

UARTPeriphID3 寄存器

表 1046。  
UARTPERIPHID3  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>配置</b> : 这些位读取值为 0x00	只读	0x00

## UART：UARTPCCELLID0 寄存器

偏移量：0fff0

### 描述

UARTPCCellID0 寄存器

表 1047。  
UARTPCELLID0  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>UARTPCELLID0</b> : 这些位读取值为0x0D	只读	0x0d

## UART: UARTPCELLID1寄存器

偏移: 0xff4

### 描述

UARTPCellID1 寄存器

表1048  
UARTPCELLID1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>UARTPCELLID1</b> : 这些位读取值为0xF0	只读	0xf0

## UART: UARTPCELLID2寄存器

偏移: 0xff8

### 描述

UARTPCellID2 寄存器

表1049  
UARTPCELLID2  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>UARTPCELLID2</b> : 这些位读取值为0x05	只读	0x05

## UART: UARTPCELLID3寄存器

偏移: 0ffc

### 描述

UARTPCellID3 寄存器

表1050  
UARTPCELLID3  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>UARTPCELLID3</b> : 这些位读取值为0xB1	只读	0xb1

## 12.2. I2C

### Synopsys 文档

Synopsys 专有。经许可使用。

I2C 是一种常用的双线接口，可用于通过时钟线 SCL 和数据线 SDA 连接设备，实现低速数据传输。

RP2350 具有两个相同的 I2C 控制器实例。每个控制器的外部引脚连接至第 9.4 节 GPIO 复用表中定义的 GPIO 引脚。复用选项提供一定的 IO 灵活性。

## 12.2.1. 功能特性

每个 I2C 控制器基于 Synopsys DW\_apb\_i2c(v2.03a) IP 配置，支持以下特性：

- 主设备或从设备（默认主模式）
- 标准模式、快速模式或快速模式增强
- 默认从设备地址为 0x055
- 主模式支持 10 位地址
- 16 单元发送缓冲区
- 16 单元接收缓冲区
- 支持 DMA 驱动
- 可产生中断

### 12.2.1.1. 标准

I2C 控制器设计遵循 2014 年 4 月发布的 I2C 总线规范，第 6.0 版。

### 12.2.1.2. 时钟

I2C 控制器中的所有时钟均连接至 clk\_sys，包括后文提及的 ic\_clk。I2C 时钟通过分频生成，由模块内部寄存器控制。

### 12.2.1.3. 输入输出端口

每个控制器必须将时钟信号 SCL 和数据信号 SDA 连接到一对 GPIO。I2C 标准要求驱动器将信号拉低，非驱动状态下信号由上拉电阻拉高。该要求适用于 SCL 和 SDA。GPIO 引脚应配置为：

- 启用上拉
- 限制转换速率
- 启用施密特触发器

#### **i 注意**

电路板上还应配置外部上拉，因为内部引脚上的上拉可能不足以将外部电路拉高。  
外部电路。

## 12.2.2. IP 配置

I2C 配置详情（每个实例完全独立）：

- 32 位 APB 访问
- 支持标准模式、快速模式及快速模式加（不支持高速模式）
- 默认从机地址为 0x055
- 主机模式或从机模式
- 默认主机模式（复位时从机模式禁用）

- 主机模式支持 10 位寻址（默认 7 位寻址）
- 16 条传输缓冲区条目
- 16 条接收缓冲区条目
- 允许主机执行重启条件（可为兼容旧设备而禁用）
- 可配置定时以调整 `TsuDAT`/`ThDAT`
- 复位后响应通用调用
- 具备 DMA 接口
- 单一中断输出
- 可配置定时以调整时钟频率
- 脉冲抑制（默认 7 个 `clk_sys` 周期）
- 从机接收数据后可响应 NACK
- 当 TX FIFO 为空时保持传输
- 保持总线，直到 RX FIFO 中有可用空间
- 从模式下重新启动检测中断
- 可选阻塞式主设备命令（默认未启用）

### 12.2.3. I2C 概述

I2C总线是一种两线串行接口，包括串行数据线 `SDA`和串行时钟线 `SCL`。这些线缆在连接至总线的设备之间传递信息。每个设备通过唯一地址识别，并可根据设备功能作为“发送器”或“接收器”运行。执行数据传输时，设备亦可被视为主设备或从设备。主设备为启动总线数据传输并生成时钟信号以允许传输的设备。此时，任何被寻址的设备均视为从设备。

#### 注意

I2C模块必须仅配置为主模式或从模式之一。不支持同时作为主设备和从设备运行。

I2C模块可运行于以下模式：

- 标准模式（数据速率为0至100 kb/s），
- 快速模式（数据速率最高可达400 kb/s），
- 快速增强模式（数据速率最高可达1000 kb/s）。

以下模式不被支持：

- 高速模式（数据速率最高可达3.4 Mb/s），
- 超高速模式（数据速率最高可达5 Mb/s）。

#### 注意

除非另有特别说明，快速模式的相关描述同样适用于快速增强模式。

只要设备连接至总线，I2C模块便能以任一模式与之通信。

此外，快速模式设备具有向下兼容性。例如，快速模式设备可以通过I2C总线系统以最高100 kb/s的速率与标准模式设备通信。然而，标准模式设备不具备向上兼容性，因其无法适应更高的传输速率，不应纳入快速模式的I2C总线系统；否则可能导致不可预测的状态。

以下设备通常使用高速模式：

- 液晶显示屏
- 高位数模数转换器
- 高容量EEPROM

这些设备通常需要传输大量数据。

大多数维护和控制应用，即I2C总线的常见用途，通常以100 kHz的标准模式和快速模式运行。任何 DW\_apb\_i2c 设备均可连接至I2C总线。每个设备都能与任何主机通信，实现双向信息传递。总线上必须至少有一个主机（例如微控制器或数字信号处理器），但也可存在多个主机，且需要它们进行所有权仲裁。本章后续将介绍多个主机和仲裁机制。该I2C模块不支持SMBus及PMBus协议（系统管理和电源管理协议）。

DW\_apb\_i2c由以下部分组成：

- 一个AMBA APB从属接口
- 一个I2C接口
- 用于维护两个接口一致性的FIFO逻辑

该组件的结构如图68所示。

图68。I2C块图



以下定义了图68中各模块的功能：

- **AMBA总线接口单元：**接收APB接口信号并将其转换为通用接口，从而使寄存器文件与总线协议无关。
- **寄存器文件：**包含配置寄存器，是与软件的接口。
- **从属状态机：**遵循从属协议并监控总线以匹配地址。
- **主控状态机：**生成主控传输所需的I2C协议。
- **时钟发生器：**计算执行以下操作所需的时序：
  - 在配置为主控时生成 SCL 时钟
  - 检查总线是否空闲
  - 生成启动（START）和停止（STOP）信号
  - 设置并保持数据
- **接收移位寄存器：**将数据接入设计并以字节格式提取。

- **发送移位寄存器**: 输出CPU提供的数据以进行I2C总线传输。
- **接收滤波器**: 检测总线上的各类事件；例如启动、停止及仲裁丢失。
- **切换器**: 在两端产生脉冲并切换，用于跨时钟域传输信号。
- **同步器**: 实现信号从一个时钟域向另一个时钟域的传输。
- **DMA接口**: 生成与中央DMA控制器的握手信号，实现无CPU干预的自动数据传输。
  
- **中断控制器**: 生成原始中断信号和中断标志，支持其设置与清除。
- **RX FIFO/TX FIFO**: 包含RX FIFO和TX FIFO寄存器组及控制器及其状态级别。

## 12.2.4. I2C 术语

本节定义I2C各部分使用的关键术语。

### 12.2.4.1. I2C总线术语

以下术语涉及I2C设备的角色及其与总线上其他I2C设备的交互。

#### 发送器

向总线发送数据的设备。发送器可以是发起数据传输的设备（主发送器），也可以是响应主机请求向总线发送数据的设备（从发送器）。

#### 接收器

从总线接收数据的设备。接收器可以是自行请求接收数据的设备（主接收器），也可以是响应主机请求接收数据的设备（从接收器）。

#### 主设备

初始化传输（START命令）、产生时钟信号SCL并终止传输（STOP命令）的组件。主设备可以是发送器或接收器。

#### 从设备

由主设备寻址的设备。从设备可以是接收器或发送器。

#### 多主设备

多个主设备能够在总线上同时共存且无冲突或数据丢失的能力。

#### 仲裁

授权同一时间仅允许一个主设备控制总线的预定义程序。有关此行为的更多信息，请参见第12.2.8节。

#### 同步

用于同步两个或多个主设备所提供时钟信号的预定义程序。有关此功能的更多信息，请参见第12.2.9节。

#### SDA

数据信号线（串行数据线）。

#### SCL

时钟信号线（串行时钟线）。

### 12.2.4.2 总线传输术语

以下术语专指发生在I2C总线上的数据传输。

#### 起始（重新起始）

数据传输从 START 或 RESTART 条件开始。当 SCL 时钟线保持高电平时，SDA 数据线的电平由高变低。此时，总线变为忙碌状态。

#### 注意

START 和 RESTART 条件在功能上完全相同。

#### STOP

数据传输由 STOP 条件终止。当 SCL 时钟线保持高电平时，SDA 数据线的电平由低变高，即发生此情况。数据传输终止后，总线再次处于空闲状态。如果产生 RESTART 条件而非 STOP 条件，总线将保持忙碌状态。

### 12.2.5. I2C 行为

DW\_apb\_i2c 可通过软件控制设置为以下模式之一：

- 仅作为 I2C 主设备，与其他 I2C 从设备通信。
- 仅作为 I2C 从设备，与一个或多个 I2C 主设备通信。

主设备负责生成时钟信号并控制数据传输。从设备负责向主设备传输或接收数据。数据确认由接收数据的设备发送，该设备可以是主设备或从设备。如前所述，I2C 协议允许多个主设备存在于 I2C 总线上，并采用仲裁程序确定总线所有权。

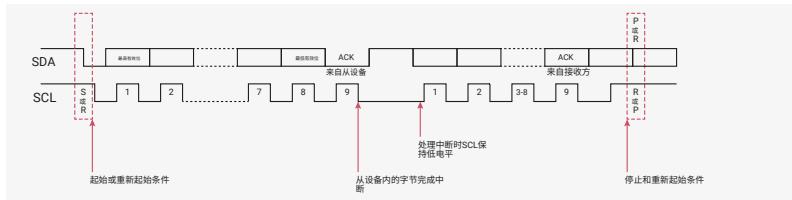
每个从设备具有由系统设计者确定的唯一地址。当主设备欲与从设备通信时：

1. 主设备发送 START/RESTART 条件，随后发送从设备地址及一个控制位 (R/W)，以确定主设备是发送数据还是接收数据。
2. 从设备随后在地址后发送确认 (ACK) 信号。

当主设备（主发送器）向从设备（从接收器）写入时，接收器获取一个字节数据。该事务持续进行，直至主设备以 STOP 条件终止传输。

当主设备从从设备读取时（主接收器），从设备（从发送器）向主设备传送一个数据字节。主设备随后通过 ACK 脉冲确认该事务。该事务持续进行，直到主设备在接收最后一个字节后未确认（发送 NACK），随后发出 STOP 条件，或在发出 RESTART 条件后寻址另一从设备，从而终止传输。该行为如图69所示。

图69。I2C总线上的数据传输



DW\_apb\_i2c 是一种同步串行接口。SDA 线为双向信号，仅在 SCL 线为低电平时发生变化，停止 (STOP)、启动 (START) 及重新启动 (RESTART) 条件除外。输出驱动器采用开漏 (open-drain) 或开集电极 (open-collector) 结构，以实现总线上的线与 (wire-AND) 功能。总线上设备数量的最大限制仅取决于最大电容规格 400 pF。数据以字节包形式传输。

实现于 DW\_apb\_i2c 中的 I2C 协议详见第 12.2.6 节。

### 12.2.5.1. 启动 (START) 和停止 (STOP) 条件的产生

作为I2C主控时，将数据写入TX FIFO会使 DW\_apb\_i2c在I2C总线上产生启动 (START) 条件。向IC\_DATA\_CMD.STOP写入1会使 DW\_apb\_i2c在I2C总线上产生停止 (STOP) 条件；若未设置该位，即使TX FIFO为空，也不会发出停止 (STOP) 条件。

当以从机模式运行时， DW\_apb\_i2c不会按照协议产生 START 和 STOP 条件。然而，如果向 DW\_apb\_i2c发出读请求，该设备将保持 SCL线为低电平，直至读取数据提供完成。这会阻塞 I2C 总线，直到将读取数据传送至从属设备 DW\_apb\_i2c，或通过向 IC\_ENABLE.ENABLE 寄存器写入 0 以禁用 DW\_apb\_i2c从机功能。

### 12.2.5.2 组合格式

DW\_apb\_i2c支持在 7 位和 10 位寻址模式下的混合读写组合格式事务。

DW\_apb\_i2c不支持混合地址格式的组合格式事务，即不支持先进行 7 位地址事务后接 10 位地址事务，或反之亦然。要启动组合格式传输，需将 IC\_CON.IC\_RESTART\_EN 设置为 1。在该值设置且作为主设备运行时，当 DW\_apb\_i2c完成 I2C 传输后，会检查 TX FIFO 并执行下一次传输。若此次传输方向与前一次传输不同，则采用组合格式发起传输。当前 I2C 传输完成且 TX FIFO 为空时：

- 检查 IC\_DATA\_CMD.STOP，且：
  - 若设置为 1，则发送 STOP 位。
  - 若设置为 0，则在下一命令写入 TX FIFO 前， SCL 保持低电平。

详情请参阅第 12.2.7 节。

## 12.2.6. I2C 协议

本节定义了 DW\_apb\_i2c所使用的协议。

### 12.2.6.1. START 和 STOP 条件

总线空闲时，总线上 SCL和 SDA信号通过外部上拉电阻被拉高。主设备欲开始总线传输时，发出**START** 条件：即在SCL为 1 时，SDA由高电平跳变至低电平。当主设备欲终止传输时，主设备会发出**STOP**条件：在SCL信号为1时，SDA信号由低电平跳变至高电平。图70展示了**START**和**STOP**条件的时序。

当数据在总线上传输时， SDA信号在 SCL 为高电平时必须保持稳定。

图70。 I2C启动  
与停止条件



### 注意

图70所示的START/STOP条件信号跳变，反映了驱动I2C总线的主设备输出信号情况。观察从设备输入端的 SDA/SCL信号时，应注意线路时延不一致可能导致 SDA/SCL时序关系出现误差。

## 12.2.6.2. 从设备寻址协议

地址格式有两种：7位和10位。

### 12.2.6.2.1. 7位地址格式

在7位地址格式中，第一个字节的高七位（位7至位1）设定从机地址，最低位（位0）定义读/写状态，如图71所示。当位0设为0时，主机向从机写入数据。当位0设为1时，主机从从机读取数据。

图71. I2C 7位地址格式



### 12.2.6.2.2. 10位地址格式

10位地址格式每个10位地址传输两个字节。

- 第一个字节中的前五位（位7至位3）表示10位地址传输。后两个位（位2:1）包含从属设备地址的位9:8。最低有效位（位0）定义读/写状态。
- 第二个字节包含从属设备地址的位7:0。

图 72 显示了 10 位地址格式：

图 72. 10 位地址格式



本表定义了特殊用途及保留的首字节地址。

表 1051.  
I2C/SMBus 首字节  
位定义

从机地址	读/写位	描述
0000 000	0	通用呼叫地址。 <a href="#">DW_apb_i2c</a> 将数据放入接收缓冲区， 并触发通用呼叫中断。
0000 000	1	起始字节。详细信息请参见第 <a href="#">12.2.6.4</a> 节。
0000 001	X	CBUS地址。 <a href="#">DW_apb_i2c</a> 忽略 这些访问。
0000 010	X	保留。

从机地址	读/写位	描述
0000 011	X	保留。
0000 1XX	X	高速主控制码（详情请参见第12.2.8节）。
1111 1XX	X	保留。
1111 0XX	X	10位从地址。
0001 000	X	SMBus主机。（不支持）
0001 100	X	SMBus警报响应地址。（不支持）
1100 001	X	SMBus设备默认地址。（不支持）

DW\_apb\_i2c不限制您使用保留地址。但若使用这些保留地址，可能会导致与I2C组件不兼容。

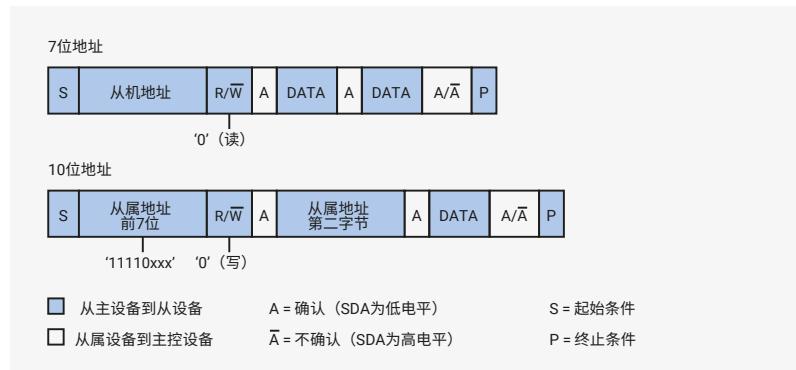
### 12.2.6.3 发送和接收协议

主控制器可在总线上发起数据发送与接收，充当主发送器或主接收器。从机响应主控制器请求，执行向总线数据发送或接收，分别充当从发送器或从接收器。

#### 12.2.6.3.1. 主发射器与从接收器

所有数据均以字节格式传输，且每次数据传输的字节数无限制。主设备发送地址和读写位或向从设备传输一字节数据后，从接收器必须响应确认信号（ACK）。若无从接收器以ACK脉冲响应，则主设备通过发出停止条件中止传输。从设备必须保持 SDA线为高电平，以便主设备能够中止传输。如果主发射器正在传输数据（如图73所示），从接收器在接收每字节数据后向主发射器发送确认脉冲。

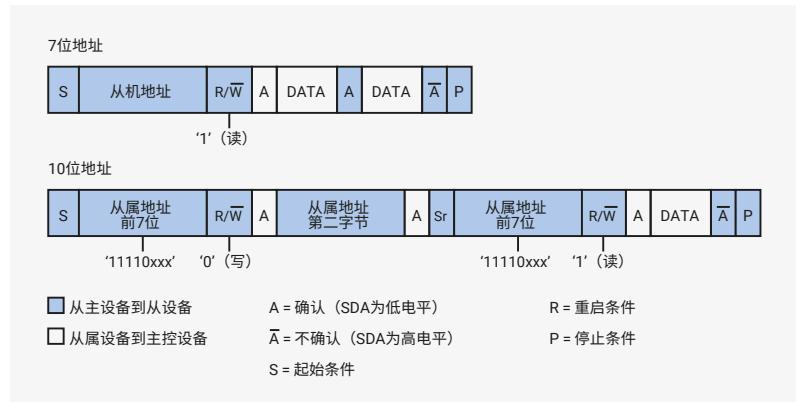
图73。I2C 主控设备-发送协议



#### 12.2.6.3.2. 主接收从发送

如图74所示，若主控设备处于接收状态，除最后一字节外，主控设备在接收每个数据字节后都会向从发送器发送确认脉冲。主接收器通过此方式告知从发送器这是最后一个字节。从发送器检测到无确认（NACK）后释放 SDA线，以便主控设备发出停止条件。

图74。I2C 主控设备-接收协议



当主设备不希望通过STOP条件释放总线时，主设备可以发送RESTART条件。

这与START条件相同，只不过发生在ACK脉冲之后。在主模式下，DW\_apb\_i2c能够通过方向不同的传输与同一从设备通信。有关DW\_apb\_i2c支持的组合格式事务的详细描述，请参见第12.2.5.2节。

### 注意

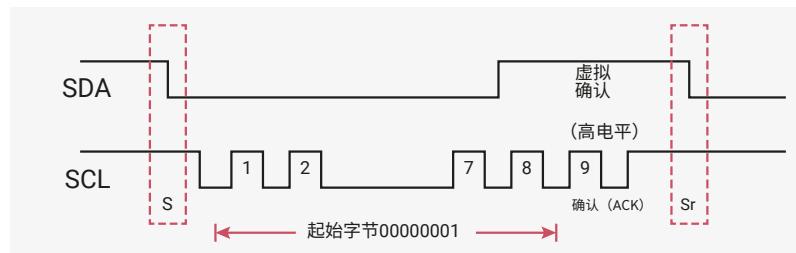
必须先完全禁用DW\_apb\_i2c，才能重新编程目标从设备地址寄存器（IC\_TAR）。

#### 12.2.6.4. START BYTE传输协议

START BYTE传输协议专为无板载专用I2C硬件模块的系统设计。当DW\_apb\_i2c作为从设备被寻址时，它总是以其支持的最高速率采样I2C总线，因此无需START BYTE传输。然而，当DW\_apb\_i2c作为主设备时，支持在每次传输开始时生成START BYTE传输，以满足从设备的需求。

该协议包括发送七个零，随后是一个一，如图75所示。这使得轮询总线的处理器能够在检测到零之前对地址阶段进行欠采样。一旦微控制器检测到零，便会从欠采样速率切换至主设备的正确速率。

图75. I2C起始字节传输



START BYTE过程如下：

1. 主设备生成START条件。
2. 主设备发送START字节（0000 0001）。
3. 主设备发送ACK时钟脉冲。（仅为符合总线上字节处理格式而设）
4. 无从设备将ACK信号拉低。
5. 主设备生成RESTART（重启）条件。

硬件接收器不响应START BYTE过程，因其使用保留地址，且在RESTART条件产生后复位。

## 12.2.7. 发送 FIFO 管理及 START、STOP 和 RESTART 生成

作为主设备操作时，DW\_apb\_i2c组件支持图76所示的TX（发送）FIFO管理模式。

### 12.2.7.1 TX FIFO管理

当TX FIFO变为空时，组件不会生成STOP信号；此时，组件保持SCL线为低电平，暂停总线，直到TX FIFO中有新条目可用。只有用户在写入IC\_DATA\_CMD寄存器的命令中明确设置第九位（停止位）时，才会生成STOP条件。图76显示了IC\_DATA\_CMD寄存器的各个位。

图76。

IC\_DATA\_CMD  
寄存器

图77说明了DW\_apb\_i2c作为主机发送器工作时，TX FIFO变为空时的行为，以及STOP条件的生成。

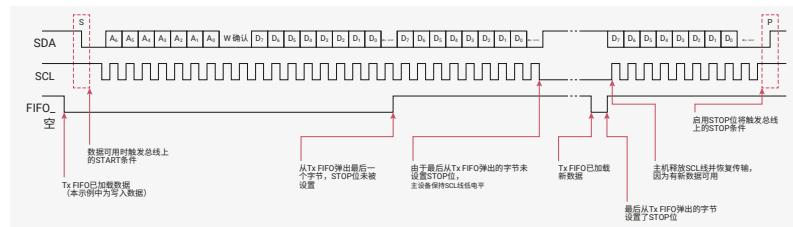
图77。主设备  
发送器 - TX FIFO  
清空/STOP  
生成

图78说明了DW\_apb\_i2c作为主接收器运行时，当TX FIFO变为空时的行为，以及STOP条件的生成。

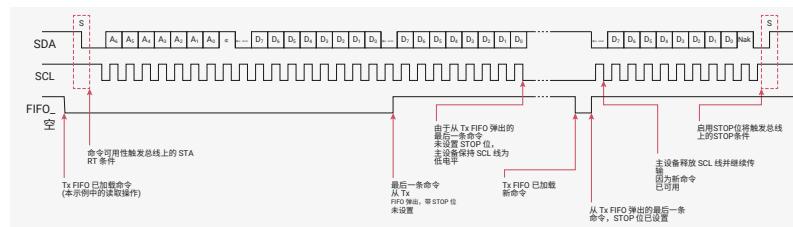
图 78。主接收器 - TX FIFO清空/  
停止信号生  
成

图 79 和图 80 展示了用户可控制 I2C 总线上 RESTART 条件生成的配置。如果 IC\_DATA\_CMD 寄存器的第 10 位（Restart）被设置且重启功能启用（IC\_RESTART\_EN=1），则在数据字节写入或读取从设备之前会生成 RESTART。如未启用重启功能，则以 STOP 后跟 START 替代 RESTART。图 79 展示作为主设备发送器时的此操作情形。

图79。主发送器 - IC\_DATA\_CMD 寄存器的重启位已置位

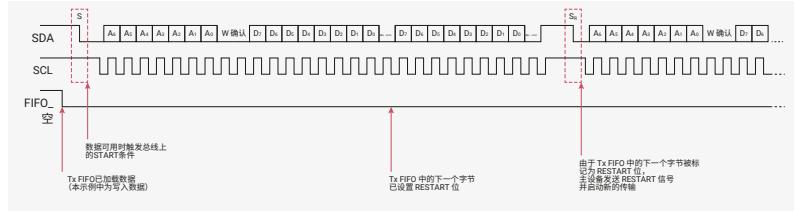


图80展示了相同情况，但操作为主设备接收器时的情形。

图80。主设备接收器 - IC\_DATA\_CMD 寄存器的重启位被设置

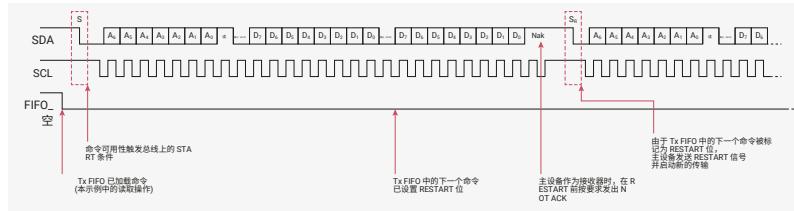


图81展示了作为主设备发送器操作时，IC\_DATA\_CMD 寄存器的停止位被设置且 TX FIFO 非空的情况。

图81。主发送器 - IC\_DATA\_CMD 停止位已置位，TX FIFO非空

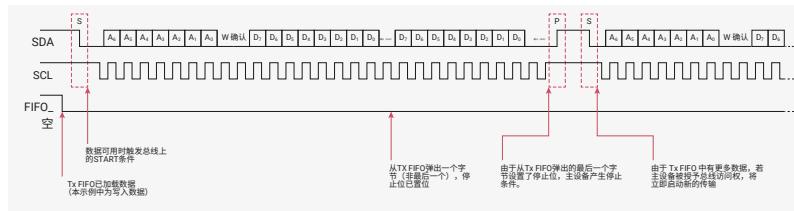


图82说明了主设备发送器的操作，其中加载到TX FIFO的第一个字节允许清空，并设置了重启位。

图82。主设备发送器—允许第一个字节清空，重启位已设置

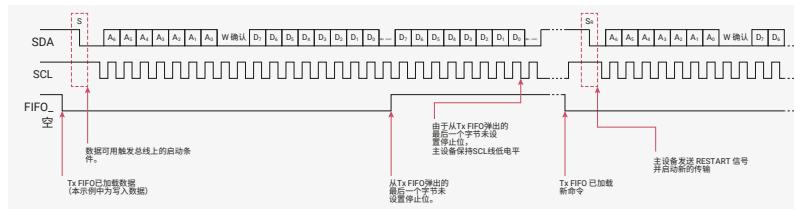


图83展示了作为主设备接收器的操作，其中IC\_DATA\_CMD寄存器的停止位被设置且TX FIFO非空。

图83。主设备接收器 - IC\_DATA\_CMD 停止位已置位/发送 FIFO非空

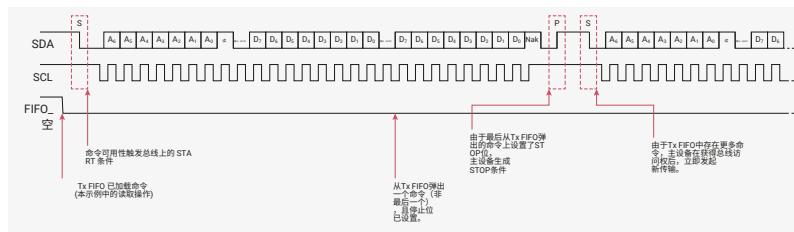
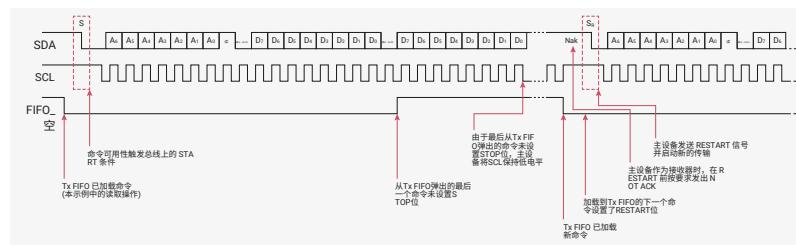


图84展示了作为主设备接收器的操作，其中在TX FIFO被允许清空后加载了首个命令，并且Restart位被设置。

图84。主设备  
接收器 - TX FIFO  
允许清空后  
加载的首个  
命令/Restart位  
已设置



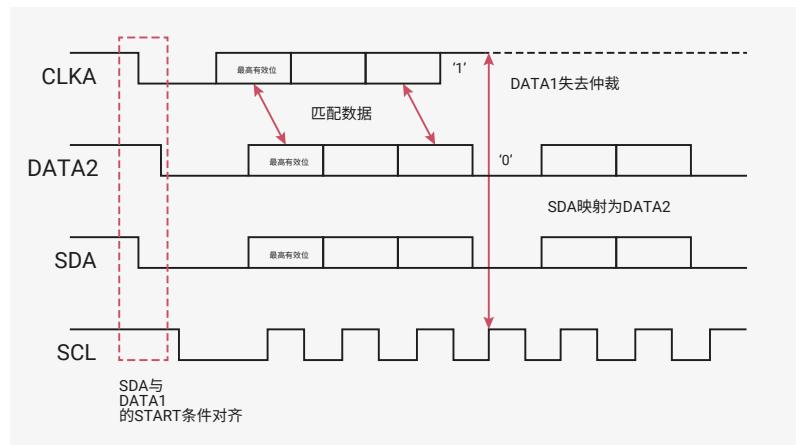
### 12.2.8. 多主仲裁

**DW\_apb\_i2c**总线协议允许多个主设备共存于同一总线。当总线上有两个主设备且两者同时通过同时生成START条件争夺总线控制权时，将执行仲裁程序。一旦主设备（例如微控制器）掌控总线，除非该主设备发送STOP条件并使总线进入空闲状态，否则其他主设备不得取得控制权。

仲裁过程发生在 **SDA**线上，同时 **SCL**线保持为1。发送1的主设备在另一主设备发送0时，失去仲裁并关闭其数据输出级。失去仲裁的主设备可继续生成时钟，直至字节传输结束。若两个主设备同时寻址同一从设备，仲裁可能延续至数据传输阶段。

检测到被另一主设备仲裁失败后，**DW\_apb\_i2c**通过禁用输出驱动停止生成 **SCL**信号。图85展示了两个主设备在总线上进行仲裁的时序。

图85。多主设  
备仲裁



总线控制由地址或主控码及竞争主控器发送的数据决定，因此总线上无中央主控器，也不存在任何优先级顺序。

下列情况之间不允许进行仲裁：

- 重启（RESTART）条件与数据位
- 停止（STOP）条件与数据位
- 重启（RESTART）条件与停止（STOP）条件

**i 注意**

从设备不参与仲裁过程。

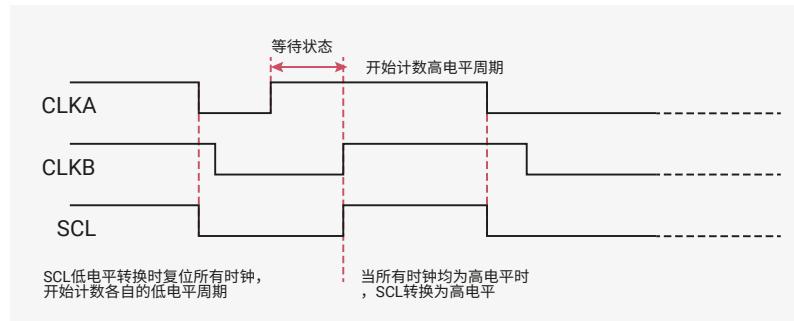
### 12.2.9. 时钟同步

当两个或多个主控器同时试图在总线上传输信息时，必须进行仲裁并同步 **SCL**时钟。所有主控器均生成各自时钟以传输消息。数据仅在 **SCL**的高电平期间有效

时钟。时钟同步通过对 **SCL** 信号的有线与（wired-AND）连接实现。当主控器将 **SCL** 时钟拉低至零时，主控器开始计数 **SCL** 时钟的低电平时间，并在下一时钟周期开始时将 **SCL** 时钟信号拉高至高电平。然而，若另一主控设备将 **SCL** 线保持为 0，则主控设备进入高等待状态，直至 **SCL** 时钟线转换为 1。

所有主控设备随后计数其高电平时间，拥有最短高电平时间的主控设备将 **SCL** 线拉低。主控设备继续计数其低电平时间，具有最长低电平时间的主控设备强制其他主控设备进入高等待状态。由此产生一个同步的 **SCL** 时钟信号，如图86所示。从设备可选择将 **SCL** 线拉低，以减缓I2C总线的时序。

图86. 多主设备  
时钟  
同步



## 12.2.10. 操作模式

本节提供有关操作模式的详细信息。

### ① 注意

仅将 **DW\_apb\_i2c** 配置为 I2C 主设备或从设备，切勿同时配置为两者。为避免此情况，切勿同时将 **IC\_CON.IC\_SLAVE\_DISABLE** 设为 0 和 **IC\_CON.MASTER\_MODE** 设为 1。

### 12.2.10.1. 从属模式操作

本节讨论从属模式的操作流程。

#### 12.2.10.1.1. 初始配置

要使用 **DW\_apb\_i2c** 作为从属设备，请执行以下步骤：

1. 通过向 **IC\_ENABLE.ENABLE** 写入 0 来禁用 **DW\_apb\_i2c**。
2. 向 **IC\_SAR** 寄存器（位 9:0）写入数据以设置从属地址。该地址即为 **DW\_apb\_i2c** 响应的地址。
3. 通过向 **IC\_CON** 寄存器写入数据以指定支持的寻址类型（设置第 3 位以选择 7 位或 10 位寻址）。  
通过向第六位（**IC\_CON.IC\_SLAVE\_DISABLE**）写入 0，且向第零位（**IC\_CON.MASTER\_MODE**）写入 0，以仅从属模式启用 **DW\_apb\_i2c**。

**i 注意**

从机与主机可以采用不同的寻址设置。例如，从机可被编程为7位寻址，主机可为10位寻址，反之亦然。

4. 通过向IC\_ENABLE.ENABLE寄存器写入1来启用 DW\_apb\_i2c。

**i 注意**

根据配置的复位值，步骤二和步骤三可能无需执行，因为复位值可直接配置。例如，若设备仅作为主机使用，无需设置从机地址，因可配置 DW\_apb\_i2c在复位后禁用从机并启用主机功能。存储的值为静态值，若 DW\_apb\_i2c被禁用，则无需重新编程。

**— 警告**

仅在I2C总线空闲时解除 DW\_apb\_i2c从机的复位。若总线正进行数据传输时解除复位，会导致用于同步 SDA和 SCL的内部同步触发器由复位状态1切换至总线上的实际值，从而可能使 SDA在 SCL为1时从1跳变至0，导致 DW\_apb\_i2c从机误判为启动条件。为避免此情况，可将 DW\_apb\_i2c配置为IC\_SLAVE\_DISABLE=1及MASTER\_MODE=1，从而在复位后禁用从机接口。然后，可以通过在内部 SDA和 SCL与总线上的值同步后，将IC\_CON[0] = 0和IC\_CON[6] = 0编程启用；这大约需要重置信号取消置位后六个 ic\_clk周期。

**12.2.10.1.2. 单字节从机发送器操作**

当总线上另一个I2C主设备寻址 DW\_apb\_i2c并请求数据时， DW\_apb\_i2c作为从机发送器响应。以下步骤随之发生：

1. 其他I2C主设备启动I2C传输，地址与 DW\_apb\_i2c的IC\_SAR寄存器中的从机地址匹配。
2. DW\_apb\_i2c应答该地址，并识别传输方向，表明其作为从机发送器身份。
3. DW\_apb\_i2c断言RD\_REQ中断（IC\_RAW\_INTR\_STAT寄存器第5位），并将SCL线保持低电平，处于等待软件响应的状态。如果RD\_REQ中断已被屏蔽，原因是  
由于IC\_INTR\_MASK.M\_RD\_REQ被设置为零，应使用硬件和/或软件定时程序指示CPU周期性读取IC\_RAW\_INTR\_STAT寄存器。
  - o 当读取显示IC\_RAW\_INTR\_STAT.RD\_REQ被设置为1时，必须视同RD\_REQ中断被触发。
  - o 软件随后必须采取措施以完成I2C传输。
  - o 所用定时间隔应约为DW\_apb\_i2c可处理的最快SCL时钟周期的10倍。例如，对于400 kb/s，定时间隔为2 5μs。

**i 注意**

推荐使用数值10，因为这大致是I2C总线上传输单字节数据所需的时间。

4. 如果在接收读请求之前TX FIFO中仍有数据， DW\_apb\_i2c将断言 TX\_ABRT中断（IC\_RAW\_INTR\_STAT寄存器的第六位）以清空TX FIFO中的旧数据。若由于IC\_INTR\_MASK.M\_TX\_ABRT被设置为零导致TX\_ABRT中断被屏蔽，则应重复使用前述定时程序读取IC\_RAW\_INTR\_STAT寄存器。

### ● 注意

由于 DW\_apb\_i2c 的 TX FIFO 在发生 TX\_ABRT 事件时被强制置于刷新/复位状态，软件必须通过读取 IC\_CLR\_TX\_ABRT 寄存器解除该状态，然后才能尝试写入 TX FIFO。详见寄存器 IC\_RAW\_INTR\_STAT。

- 读取值中若第六位（R\_TX\_ABRT）被置为1，须视同 TX\_ABRT 中断被触发。
  - 软件无需再采取其他操作。
  - 所用的时间间隔应与前述步骤中对应 IC\_RAW\_INTR\_STAT.RD\_REQ 寄存器的时间间隔相似。
5. 软件通过将第8位写为0，向 IC\_DATA\_CMD 寄存器写入需发送的数据。
  6. 软件必须在继续操作前，清除 IC\_RAW\_INTR\_STAT 寄存器中第5位和第6位的 RD\_REQ 及 TX\_ABRT 中断。若 RD\_REQ 或 TX\_ABRT 中断已被屏蔽，则在读取到 R\_RD\_REQ 或 R\_TX\_ABRT 位为1时，IC\_RAW\_INTR\_STAT 寄存器已被清除。
  7. DW\_apb\_i2c 释放了 SCL 线并传输该字节。
  8. 主设备可通过发出重启（RESTART）条件保持 I2C 总线，或通过发出停止（STOP）条件释放总线。

### ● 注意

超高速（Ultra-Fast）模式下不适用单字节从设备发送操作，因为该模式不支持读传输。

#### 12.2.10.1.3. 单字节从设备接收操作

当总线上另一个 I2C 主设备对 DW\_apb\_i2c 寻址并发送数据时，DW\_apb\_i2c 作为从设备接收器，执行以下步骤：

1. 另一 I2C 主设备使用与 IC\_SAR 寄存器中 DW\_apb\_i2c 从设备地址匹配的地址启动 I2C 传输。
2. DW\_apb\_i2c 确认所发送的地址并识别传输方向，表明 DW\_apb\_i2c 正在作为从设备接收器工作。
3. DW\_apb\_i2c 接收传输的字节，并将其存入接收缓冲区。

### ● 注意

若在推送字节时 Rx（接收） FIFO 已满，则 DW\_apb\_i2c 从机会将 I2CSCL 线拉低，直至 Rx FIFO 具备空间，随后继续处理下一读请求。

4. DW\_apb\_i2c 会触发 RX\_FULL 中断 IC\_RAW\_INTR\_STAT.RX\_FULL。若因设置 IC\_INTR\_MASK.M\_RX\_FULL 为零或把 IC\_TX\_TL 设为大于零值而屏蔽 RX\_FULL 中断，应实施一个定时程序（参见第 12.2.10.1.2 节），以对 IC\_STATUS 寄存器进行周期性读取。该定时程序应将 IC\_STATUS 寄存器中第3位（RFNE）为1的读取视同 RX\_FULL 中断。
5. 软件可从 IC\_DATA\_CMD 寄存器（位7:0）读取该字节。
6. 另一个主设备可通过发出重启（RESTART）条件持有 I2C 总线，或通过发出停止（STOP）条件释放总线。

#### 12.2.10.1.4. 用于批量传输的从设备传输操作

在标准I2C协议中，所有事务均为单字节传输；程序员通过向从设备的TX FIFO写入一个字节来响应远程主设备的读请求。当从设备（作为发送方）接收到读请求（RD\_REQ）来自远程主设备（作为接收方）时，至少应在从设备发送方的TX FIFO中放入一条数据。

DW\_apb\_i2c处理TX FIFO中的更多数据，使随后的读请求可在不触发中断的情况下获取数据。此举消除了中断之间产生的延迟。此模式仅在 DW\_apb\_i2c作为从设备发送方时发生。若远程主设备确认了从设备发送方发送的数据且TX FIFO中无数据，DW\_apb\_i2c将在触发读请求中断（RD\_REQ）时将I2CSCL线保持为低电平，等待数据写入TX FIFO。

如果通过将 IC\_INTR\_STAT.R\_RD\_REQ 设置为零屏蔽了 RD\_REQ中断，则应使用定时例程周期性地读取 IC\_RAW\_INTR\_STAT 寄存器。读取 IC\_RAW\_INTR\_STAT 时若返回第5位（RD\_REQ）为1，必须将其视为 RD\_REQ中断。该定时例程类似于第12.2.10.1.2节所述。

RD\_REQ中断在发生读请求时触发。退出中断服务例程（ISR）时，务必清除此中断。ISR 允许向 TX FIFO 写入一个或多个字节。主设备可在传输结束时通过确认最后一个字节来请求更多数据。在此情况下，从设备必须再次触发RD\_REQ中断。

如果已知远端主设备请求的数据包长度为 n字节，可向 TX FIFO 写入 n字节。随后，当其他主设备访问 DW\_apb\_i2c并请求数据时，远端主设备将接收到连续的数据流。这是因为 DW\_apb\_i2c从属设备只要远程主设备确认所发送的数据且TX FIFO 中有数据，就会继续发送数据。无需将 SCL线保持低电平，也无需再次发出 RD\_REQ命令。

如果远程主设备未从TX FIFO中读取所有字节， DW\_apb\_i2c将按以下程序忽略多余字节：

- DW\_apb\_i2c会清空TX FIFO。
- DW\_apb\_i2c将产生传输中止（TX\_ABRT）事件。

当预期应答（ACK/NACK）时，如果收到NACK，表明远程主设备已获取所有所需数据。此时，奴隶状态机内部会触发标志以清除TX FIFO中的剩余数据。该标志会传递至FIFO所在的处理器总线时钟域，届时TX FIFO内容将被清空。

#### 12.2.10.2 主控模式操作

本节讨论主控模式的操作流程。

##### 12.2.10.2.1 初始配置

要将 DW\_apb\_i2c作为主控使用，请执行以下步骤：

1. 通过向 IC\_ENABLE.ENABLE 寄存器写入 0，禁用DW\_apb\_i2c。
2. 写入 IC\_CON 寄存器以设置支持的最大速率模式（位 2:1）及 DW\_apb\_i2c 主控发起传输所需的速率，支持 7 位或 10 位寻址（位 4）。确保第 6 位（IC\_SLAVE\_DISABLE）写入 1，第 0 位（MASTER\_MODE）写入 1。

**● 注意**

从机与主机可以采用不同的寻址设置。例如，从机可被编程为7位寻址，主机可为10位寻址，反之亦然。

3. 将目标 I2C 设备地址写入 IC\_TAR 寄存器的位 9:0。该寄存器还决定 I2C 是否执行通用调用（General Call）或启动字节命令（START BYTE）。
4. 通过向 IC\_ENABLE.ENABLE 写入 1，启用 DW\_apb\_i2c。
5. 将传输方向和要发送的数据写入 IC\_DATA\_CMD 寄存器。此步骤在 DW\_apb\_i2c 上产生 START 条件和地址字节。一旦启用 DW\_apb\_i2c 且 TX FIFO 中有数据，DW\_apb\_i2c 即开始读取数据。

**● 注意**

若在启用 DW\_apb\_i2c 之前写入 IC\_DATA\_CMD 寄存器，数据和命令将丢失：DW\_apb\_i2c 被禁用时缓冲区保持清空。

存储的值为静态值，除传输方向和数据外，DW\_apb\_i2c 禁用时无需重新编程。因此，若已配置复位值，可能无需执行第二、第三、第四及第五步。

#### 12.2.10.2.2. 主控发送与主控接收

DW\_apb\_i2c 支持读写操作的动态切换。为了传输数据，请将数据写入 I2C RX/TX 数据缓冲区及命令寄存器（IC\_DATA\_CMD）的低字节。针对 I2C 写操作，请将 CMD 位[8]写入零。随后，为发出读取命令，请将 CMD 位写入一，并将无关数据写入 IC\_DATA\_CMD 寄存器的低字节。只要 TX FIFO 中存在命令，DW\_apb\_i2c 主控器将持续发起传输。若 TX FIFO 变为空，主控器将根据 IC\_DATA\_CMD 的值执行以下任一操作：

- 若设置为一，当前传输完成后将发出 STOP 条件。
- 若设置为零，则保持 SCL 线为低电平，直至下一命令写入 TX FIFO。

详情请参阅第 12.2.7 节。

#### 12.2.10.3. 禁用 DW\_apb\_i2c

IC\_ENABLE\_STATUS 寄存器能够使软件明确判断 I2C 硬件何时已完全关闭。

**● 注意**

早期版本的 DW\_apb\_i2c 要求程序员监控两个寄存器（IC\_STATUS 和 IC\_RAW\_INTR\_STAT）。RP2350 仅要求程序员监控 IC\_ENABLE\_STATUS。

要关闭 I2C 硬件，请向 IC\_ENABLE.ENABLE 写入零。DW\_apb\_i2c 主控仅在去使能时，当前处理的命令 STOP 位被置为一时才允许禁用。若在处理未设置 STOP 位的命令时尝试禁用 DW\_apb\_i2c 主控，DW\_apb\_i2c 将继续保持激活状态，并保持 SCL 线低电平，直到 TX FIFO 收到新命令。

若在 DW\_apb\_i2c 主控处理未设置 STOP 位的命令时需要释放 I2C 总线并禁用 DW\_apb\_i2c，必须发出 ABORT 请求。

### 12.2.10.3.1. 操作流程

1. 定义定时器间隔 ( $t_{i2c,poll}$ )，其值等于系统中使用且由 `DW_apb_i2c` 支持的最高I2C传输速率对应信号周期的10倍。例如，若最高I2C传输模式为400 kb/s，则  $t_{i2c,poll}$  为25μs。
2. 定义最大超时参数 `MAX_T_POLL_COUNT`，若任何重复轮询操作超过该最大值，则报告错误。
3. 执行阻塞线程、进程或函数，在软件中阻止任何新的I2C主机事务启动，但允许所有挂起传输完成。

**注意**

若 `DW_apb_i2c` 被配置为仅作为I2C从机运行，则此步骤可省略。

1. 将变量 `POLL_COUNT` 初始化为零。
2. 将 `IC_ENABLE` 寄存器的第0位清零。
3. 读取 `IC_ENABLE_STATUS` 寄存器并检测 `IC_EN` 位（第0位）。将 `POLL_COUNT` 增加一。如果 `POLL_COUNT >= MAX_T_POLL_COUNT`，则退出并返回相应的错误代码。
4. 如果 `IC_ENABLE_STATUS[0]` 为 1，则睡眠  $t_{i2c,poll}$  并返回上一步。否则，退出并返回相应的成功代码。

### 12.2.10.4. 中止 I2C 传输

`IC_ENABLE` 寄存器的 `ABORT` 控制位允许软件在完成 TX FIFO 中发出的传输命令之前释放 I2C 总线。响应 `ABORT` 请求时，控制器将在 I2C 总线上发出 STOP 条件，随后清空 TX FIFO。仅允许在主控模式下中止传输。

### 12.2.10.4.1. 操作流程

1. 停止向 TX FIFO (`IC_DATA_CMD`) 添加新命令。
2. 在 DMA 模式下，将 `TDMAE` 设为零以禁用发送 DMA。
3. 将 `IC_ENABLE.ABORT` 设为 1。
4. 等待 `M_TX_ABRT` 中断。
5. 读取 `IC_TX_ABRT_SOURCE` 寄存器以识别源为 `ABRT_USER_ABRT`。

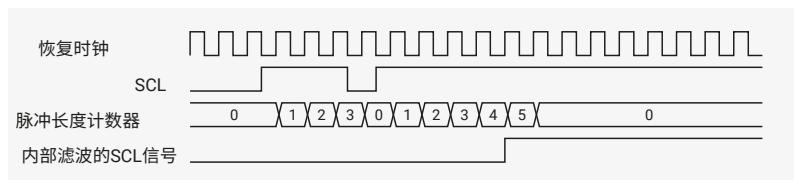
## 12.2.11. 突波抑制

`DW_apb_i2c` 包含可编程尖峰抑制逻辑，符合 I2C 总线规范对于 SS/FS 模式的要求。该逻辑基于计数器监测输入信号（SCL 和 SDA），验证其在预定的 `ic_clk` 周期内是否保持稳定，随后进行内部采样。每个信号（SCL 和 SDA）配备独立的计数器。用户可编程设置 `ic_clk` 周期数。该数值应结合 `ic_clk` 频率及相关尖峰长度规范进行设置。每当输入信号值发生变化时，相应计数器即开始计数。根据输入信号的行为，将发生以下一种情况：

- 输入信号保持不变，直至计数器达到设定的计数上限。当此情况发生时，计数器复位并停止，信号的内部版本更新为输入值。
- 输入信号在计数器达到计数限制值之前再次变化。当此情况发生时，计数器复位并停止，但信号的内部版本不会更新。

图87的时序图展示了上述行为。

图87。脉冲抑制示例



### 注意

在 `SCL` 输入端设有两级同步器。为简化起见，图87的时序图未包含此同步延迟。

I2C总线规范根据工作模式对最大脉冲长度有不同要求（SS和FS模式为50 ns）。寄存器IC\_FS\_SPKLEN存储SS和FS模式下的最大脉冲长度。

该寄存器宽8位，可通过APB接口进行读写。但仅当 `DW_apb_i2c` 被禁用时，才可写入此寄存器。这些寄存器可编程的最小值为一；

尝试写入小于一的值时，实际写入值将为一。

这些寄存器的默认值基于 `ic_clk` 周期为100纳秒，因此应根据RP2350上实际使用的`clk_sys`周期进行更新。

### 注意

- 由于IC\_FS\_SPKLEN寄存器可编程的最小值为一，针对低频率的 `ic_clk`，峰值长度可能会超过规格限制。考虑一个简单示例，10 MHz（周期100纳秒）的 `ic_clk`；在此情况下，可编程的最小峰值长度为100纳秒，意味着峰值长度不超过此值的峰将被抑制。
  - 标准同步逻辑（由两个串联触发器组成）位于峰值抑制逻辑之前，且其运行不受峰值长度寄存器内容或峰值抑制逻辑操作的影响；同步与峰值抑制两种操作完全独立。
- 由于 `SCL` 和 `SDA` 输入与 `ic_clk` 异步，因此采样这些信号时存在一个 `ic_clk` 周期的不确定性。根据它们相对于 `ic_clk` 上升沿的发生时间，相同长度的尖峰在采样后可能表现出一个 `ic_clk` 周期的差异。
- 尖峰抑制是对称的；从零到一以及从一到零的转换行为完全一致。

## 12.2.12. 快速模式加速操作

在快速模式加（fast mode plus）中，`DW_apb_i2c` 支持高达1000 kb/s的快速模式操作。在启动任何数据传输前，须执行以下步骤以启用`DW_apb_i2c`的快速模式加操作：

- 设置 `ic_clk` 频率不低于32 MHz（参见第12.2.14.2.1节）。
- 将IC\_CON寄存器的[2:1]位设置为2`b10`，以启用快速模式或快速模式加。
- 配置 IC\_FS\_SCL\_LCNT 和 IC\_FS\_SCL\_HCNT 寄存器以满足快速模式加速 `SCL` 的要求（参见第12.2.14节）。
- 配置 IC\_FS\_SPKLEN 寄存器以抑制最长50纳秒的脉冲尖峰。
- 配置 IC\_SDA\_SETUP 寄存器以满足最小数据建立时间（tSU; DAT）。

## 12.2.13. 总线清除功能

`DW_apb_i2c` 支持总线清除功能，该功能能在时钟线或数据线异常保持低电平时，优雅地恢复数据线SDA和时钟线SCL。

### 12.2.13.1. SDA 线保持低电平

如 SDA线保持低电平，主设备将执行图88和图89所示操作以实现恢复：

1. 主设备发送最多九个时钟脉冲，在这九个脉冲内恢复总线的低电平状态。
  - 时钟脉冲数量将根据从设备剩余待发送位数而定。由于最大位数为九，主设备发送最多九个时钟脉冲，并允许从设备进行恢复。
  - 主设备尝试在 SDA线上断言逻辑1，并检查 SDA线是否已恢复。若 SDA线未恢复，主设备将继续发送最多九个 SCL时钟信号。
2. 若 SDA线在九个时钟脉冲内恢复，主设备将发送停止信号以释放总线。
3. 若第九个时钟脉冲后 SDA线仍未恢复，必须对系统进行硬件复位。

图88. SDA  
通过9个 SCL时钟恢复  
时钟

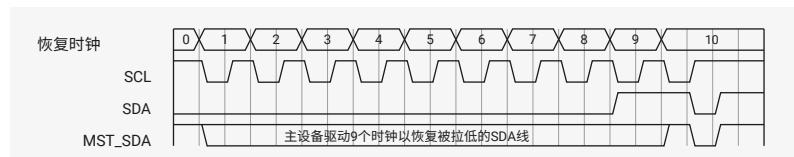
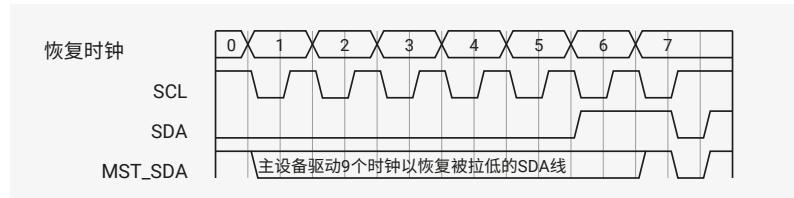


图89. SDA  
通过6个 SCL时钟恢复  
时钟



### 12.2.13.2. SCL 线被拉低

在极少数情况下（由电路电气故障导致），若时钟线（SCL）被拉低，暂无有效方法解决该问题。应使用硬件复位信号重置总线。

### 12.2.14. IC\_CLK 频率配置

当 DW\_apb\_i2c被配置为标准模式（SS）、快速模式（FS）或快速模式增强（FM+）时，必须在任何I2C总线传输开始之前设置 \*CNT寄存器，以确保正确的I/O时序。相关的 \*CNT寄存器如下：

- [IC\\_SS\\_SCL\\_HCNT](#)
- [IC\\_SS\\_SCL\\_LCNT](#)
- [IC\\_FS\\_SCL\\_HCNT](#)
- [IC\\_FS\\_SCL\\_LCNT](#)

#### i 注意

tBUF时序以及START、STOP和RESTART寄存器的建立/保持时间采用对应速度模式下的 \*HCNT/\*LCNT寄存器设置。

### 注意

若 DW\_apb\_i2c 仅被使能为 I2C 从设备，则无需配置任何 \*CNT 寄存器，因为这些寄存器仅用于确定作为 I2C 主设备时的 SCL 时序要求。

表1052列示了I2C时序参数基于 \*CNT编程寄存器的推导。

表1052。从 \*CNT 寄存器推导 I2C 定时参数

时序参数	符号	标准速率	高速 / 高速增强
SCL 时钟的低电平周期	tLOW	IC_SS_SCL_LCNT	IC_FS_SCL_LCNT
SCL 时钟的高电平周期	tHIGH	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT
重复 START 条件的建立时间	tSU;STA	IC_SS_SCL_LCNT	IC_FS_SCL_HCNT
保持时间（重复）START 条件	tHD;STA	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT
STOP 条件的建立时间	tSU;STO	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT
STOP 和 START 条件之间的总线空闲时间	tBUF	IC_SS_SCL_LCNT	IC_FS_SCL_LCNT
尖峰时长	tSP	IC_FS_SPKLEN	IC_FS_SPKLEN
数据保持时间	tHD;DAT	IC_SDA_HOLD	IC_SDA_HOLD
数据建立时间	tSU;DAT	IC_SDA_SETUP	IC_SDA_SETUP

#### 12.2.14.1. SS、FS 和 FM+ 模式下的最小高低计数

当 DW\_apb\_i2c 作为 I2C 主设备进行发送和接收传输时：

- IC\_SS\_SCL\_LCNT 和 IC\_FS\_SCL\_LCNT 寄存器值必须大于 IC\_FS\_SPKLEN + 7。
- IC\_SS\_SCL\_HCNT 和 IC\_FS\_SCL\_HCNT 寄存器值必须大于 IC\_FS\_SPKLEN + 5。

关于 DW\_apb\_i2c 高低计数的详细说明如下：

- IC\_\*\_SPKLEN + 7 作为 \*\_LCNT 寄存器的最小值，是因为 DW\_apb\_i2c 在 SCL 负沿后驱动 SDA 所需的时间。
- IC\_\*\_SPKLEN + 5 作为 \*\_HCNT 寄存器的最小值，是由于 DW\_apb\_i2c 在 SCL 高电平期间采样 SDA 所需的时间。
  - DW\_apb\_i2c 在所编程的 \*\_LCNT 值上增加一个周期，以生成 SCL 时钟的低电平周期；这是由于 SCL 低电平的计数逻辑计数至 (\*\_LCNT + 1)。
  - DW\_apb\_i2c 在所编程的 \*\_HCNT 值上增加 IC\_\*\_SPKLEN + 7 个周期，以生成 SCL 时钟的高电平周期，原因如下：
    - SCL 高电平的计数逻辑计数至 (\*\_HCNT + 1)。
    - 施加于 SCL 线的数字滤波引入了 SPKLEN + 2 个 ic\_clk 周期的延迟，其中 SPKLEN 为 当组件处于 SS 或 FS 工作模式时，该值为 IC\_FS\_SPKLEN。
    - 每当 DW\_apb\_i2c 将 SCL 驱动由高电平拉至低电平（即完成 SCL 高电平时间）时，内部逻辑会产生三个 ic\_clk 周期的延迟。因此，DW\_apb\_i2c 所能实现的最小 SCL 低电平时间为九个 ic\_clk 周期 (7 + 1 + 1)，而最小 SCL 高电平时间为十三个 ic\_clk 周期 (6 + 1 + 3 + 3)。

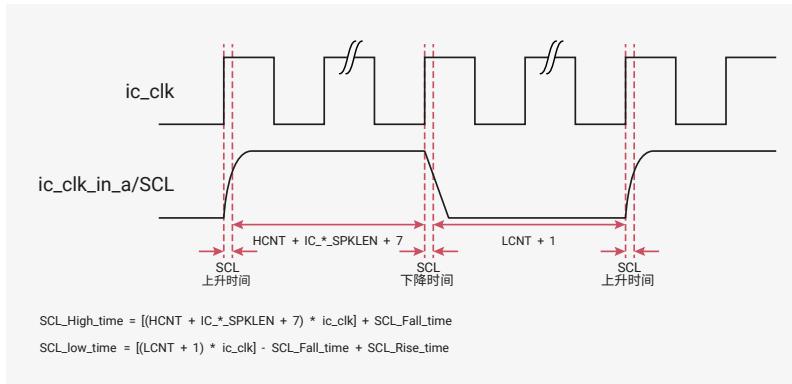
### 注意

DW\_apb\_i2c主设备生成的SCL总高电平时间和低电平时间还受到SCL信号线上升和下降时间的影响，如图90的示意图和公式所示。SCL上升时间和下降时间参数因外部因素而异，包括：

- IO驱动器的特性
- 上拉电阻的阻值
- SCL信号线上的总电容

以上特性均超出 DW\_apb\_i2c 的控制范围。

图90。SCL上升时间和下降时间对输出信号的影响  
SCL



### 12.2.14.2. 最小 IC\_CLK 频率

本节说明了 DW\_apb\_i2c 在各速率模式下支持的最小 ic\_clk 频率及对应的高计数和低计数值。在从属模式下，必须编程设置 IC\_SDA\_HOLD (Thd;dat) 和 IC\_SDA\_SETUP (Tsu:dat) 以满足 I2C 协议的时序要求。以下示例适用于将 IC\_FS\_SPKLEN 编程为二的情况。

#### 12.2.14.2.1. 标准模式 (SM)、快速模式 (FM) 及快速模式增强版 (FM+)

本节详细说明如何推导标准模式和快速模式下 DW\_apb\_i2c 的最小 ic\_clk 值。虽然以下方法展示了如何进行快速模式的计算，但您也可以使用相同方法计算标准模式和快速模式加。

### 注意

以下计算未考虑 SCL\_Rise\_time 和 SCL\_Fall\_time。

快速模式下最小 DW\_apb\_i2c ic\_clk 值的给定条件和计算：

- 快速模式的数据速率为 400 kb/s；意味着 SCL 周期为  $1/400 \text{ kHz} = 2.5 \mu\text{s}$
- 将最小 hcnt 值设为 14，作为初始值；  $\text{IC\_HCNT\_FS} = 14$
- 协议规定的最小 SCL 高电平和低电平时间：
  - $\text{MIN\_SCL\_LOWtime\_FS} = 1300 \text{ ns}$
  - $\text{MIN\_SCL\_HIGHtime\_FS} = 600 \text{ ns}$

推导方程：

$$\text{SCL\_PERIOD\_FS} / (\text{IC\_HCNT\_FS} + \text{IC\_LCNT\_FS}) = \text{IC\_CLK\_PERIOD}$$

$$\text{IC\_LCNT\_FS} \times \text{IC\_CLK\_PERIOD} = \text{MIN\_SCL\_LOWtime\_FS}$$

综上所述，前述方程得出以下结果：

$$\text{IC\_LCNT\_FS} \times (\text{SCL\_PERIOD\_FS} / (\text{IC\_LCNT\_FS} + \text{IC\_HCNT\_FS})) = \text{MIN\_SCL\_LOWtime\_FS}$$

求解  $\text{IC\_LCNT\_FS}$ ：

$$\text{IC\_LCNT\_FS} \times (2.5\mu\text{s} / (\text{IC\_LCNT\_FS} + 14)) = 1.3 \mu\text{s}$$

上述方程得出：

$$\text{IC\_LCNT\_FS} = \text{roundup}(15.166) = 16$$

计算结果为  $\text{IC\_LCNT\_FS} = 16$  和  $\text{IC\_HCNT\_FS} = 14$ ，得到的  $\text{ic\_clk}$  值为：

$$2.5\mu\text{s} / (16 + 14) = 83.3\text{ns} = 12 \text{ MHz}$$

测试验证表明已满足协议要求。

表 1053 列出了所有模式下高低计数值对应的最小  $\text{ic\_clk}$  值。

表 1053。  $\text{ic\_clk}$  与高计数和低计数的关系

速度模式	$\text{ic\_clkfreq}$ (MHz)	$\text{IC\_*_SPKLEN}$ 的最小值	SCL 低电平时间以‘ $\text{ic\_clk}$ ’计数	SCL 低电平程序值	SCL 低电平时间	SCL 在 ‘ $\text{ic\_clk}$ ’时钟周期中的高电平时间	SCL 高电平程序值	SCL 高电平时间
SS	2.7	1	13	12	4.7μs	14	6	5.2μs
FS	12.0	1	16	15	1.33μs	14	6	1.16μs
FM+	32	2	16	15	500 ns	16	7	500 ns

- $\text{IC\_*_SCL\_LCNT}$  和  $\text{IC\_*_SCL\_HCNT}$  寄存器依据表 1053 中的 SCL 低电平及高电平编程值进行编程，该值分别通过 SCL 低计数减一及 SCL 高计数减八计算得出。表 1053 中的数值基于  $\text{IC\_SDA\_RX\_HOLD} = 0$ 。最大  $\text{IC\_SDA\_RX\_HOLD}$  值取决于主模式下的  $\text{IC\_*_CNT}$  寄存器。

- 为考虑 RC 时序计算 HCNT 和 LCNT，请使用以下公式：

$$\text{IC\_HCNT\_*} = [(HCNT + IC\_*_SPKLEN + 7) * ic\_clk] + SCL\_Fall\_time$$

$$\text{IC\_LCNT\_*} = [(LCNT + 1) * ic\_clk] - SCL\_Fall\_time + SCL\_Rise\_time$$

### 12.2.14.3. 高电平计数与低电平计数的计算

以下计算展示了如何计算 DW\_apb\_i2c 各速率模式下的 SCL 高电平和低电平计数。为了确保计算正确，所使用的  $\text{ic\_clk}$  频率不得低于表 1053 中规定的最低  $\text{ic\_clk}$  频率。

默认的  $\text{ic\_clk}$  周期值设为 100 ns，因此基于该时钟，为每种速度模式计算了默认的 SCL 高电平和低电平计数值。

上述数值均基于该时钟模式。这些数值需依据以下指导原则进行更新。

计算设定正确SCL时钟高低电平时间所需适当ic\_clk信号数量的公式如下：

```
IC_xCNT = (ROUNDUP(MIN_SCL_xxxtime*OSCFREQ, 0))

MIN_SCL_HIGHTime = 最小高电平周期
MIN_SCL_HIGHTime = 100 kb/s时为4000 ns, 40
 0 kb/s时为600 ns, 10
 00 kb/s时为260 ns,

MIN_SCL_LOWtime = 最小低电平周期
MIN_SCL_LOWtime = 100 kb/s时为4700 ns, 400
 kb/s时为1300 ns, 100
 0 kb/s时为500 ns,

OSCFREQ = ic_clk 时钟频率 (Hz) 。
```

例如：

```
OSCFREQ = 100MHz
I2Cmode = 快速, 400kb/s
MIN_SCL_HIGHTime = 600ns。
MIN_SCL_LOWtime = 1300ns.

IC_xCNT = (ROUNDUP(MIN_SCL_HIGH_LOWtime*OSCFREQ, 0))

IC_HCNT = (ROUNDUP(600ns * 100MHz, 0))
IC_HCNTSCL 周期 = 60
IC_LCNT = (ROUNDUP(1300ns * 100MHz, 0))
IC_LCNTSCL 周期 = 130
实际 MIN_SCL_HIGHTime = 60 * (1/100MHz) = 600ns
实际 MIN_SCL_LOWtime = 130 * (1/100MHz) = 1300ns
```

## 12.2.15. DMA 控制器接口

该 DW\_apb\_i2c 具备内置 DMA 功能；其具有与 DMA 控制器的握手接口，用以请求和控制传输。APB 总线用于执行与 DMA 之间的数据传输。由于数据速率相对较低，DMA 传输采用单次访问。

### 12.2.15.1. 启用DMA控制器接口

要在 DW\_apb\_i2c 上启用 DMA 控制器接口，必须写入 DMA 控制寄存器 (IC\_DMA\_CR)。

将 1 写入 IC\_DMA\_CR 寄存器的 TDMAE 位字段时，即启用 DW\_apb\_i2c 的发送握手接口。

将 1 写入 IC\_DMA\_CR 寄存器的 RDMAE 位字段时，即启用 DW\_apb\_i2c 的接收握手接口。

### 12.2.15.2. 操作概述

DMA 控制器被编程为处理由 DW\_apb\_i2c 传输或接收的数据项数量（传输计数）。

传输被分解为总线上的单次传输，每次传输由 DW\_apb\_i2c 发起请求。

例如，传输计数被编程为四。DMA传输由四次单独事务组成的一系列动作构成。如果 `DW_apb_i2c` 向该通道发出传输请求，则单个数据项会写入 `DW_apb_i2c` 的 TX FIFO。同样地，如果 `DW_apb_i2c` 向该通道发出接收请求，则单个数据项会从 `DW_apb_i2c` 的 RX FIFO 读取。在全部四个数据项完成写入或读取之前，必须向该 DMA 通道发出四次独立请求。

### 12.2.15.3. 水位线级别

在 `DW_apb_i2c` 中，用以设置 DMA 突发允许的水位线寄存器无需做任何非复位值设置。具体而言，`IC_DMA_TDLR` 和 `IC_DMA_RDLR` 可保持为复位值零。这是因为 I2C 相较于系统带宽传输速率较低，因此只需单次传输。由于 DMA 控制器通常在系统总线上拥有最高优先级，传输得以快速完成。

### 12.2.16. 中断寄存器操作

表1054列出了 `DW_apb_i2c` 中断寄存器的操作方式及其设置与清除方法。部分位由硬件设置并由软件清除，另一些位则由硬件设置和清除。

表1054。中断寄存器的设置与清除

中断位域	硬件设置／软件清除	硬件设置与清除
RESTART_DET	是	否
GEN_CALL	是	否
START_DET	是	否
STOP_DET	是	否
活动	是	否
RX_DONE	是	否
TX_ABRT	是	否
RD_REQ	是	否
TX_EMPTY	否	是
TX_OVER	是	否
RX_FULL	否	是
RX_OVER	是	否
RX_UNDER	是	否

### 12.2.17. 寄存器列表

I2C0 和 I2C1 寄存器的基地址分别为 `0x40090000` 和 `0x40098000`（在 SDK 中定义为 `I2C0_BASE` 和 `I2C1_BASE`）。

**注意**

您可能会在I2C寄存器描述中看到对配置常量的引用；这些为固定值，在硬件设计阶段确定。其完整列表可在[pic-sdk GitHub仓库中的 i2c.h文件](#)中找到。

表1055。I2C  
寄存器列表

偏移量	名称	说明
0x00	IC_CON	I2C 控制寄存器
0x04	IC_TAR	I2C 目标地址寄存器
0x08	IC_SAR	I2C 从地址寄存器
0x10	IC_DATA_CMD	I2C 接收/发送数据缓冲区及命令寄存器
0x14	IC_SS_SCL_HCNT	标准速率 I2C 时钟 SCL 高电平计数寄存器
0x18	IC_SS_SCL_LCNT	标准速率 I2C 时钟 SCL 低电平计数寄存器
0x1c	IC_FS_SCL_HCNT	快速模式或快速模式 Plus I2C 时钟 SCL 高电平计数寄存器
0x20	IC_FS_SCL_LCNT	快速模式或快速模式 Plus I2C 时钟 SCL 低电平计数寄存器
0x2c	IC_INTR_STAT	I2C 中断状态寄存器
0x30	IC_INTR_MASK	I2C 中断掩码寄存器
0x34	IC_RAW_INTR_STAT	I2C 原始中断状态寄存器
0x38	IC_RX_TL	I2C 接收 FIFO 阈值寄存器
0x3c	IC_TX_TL	I2C 发送 FIFO 阈值寄存器
0x40	IC_CLR_INTR	清除联合及单独中断寄存器
0x44	IC_CLR_RX_UNDER	清除 RX_UNDER 中断寄存器
0x48	IC_CLR_RX_OVER	清除 RX_OVER 中断寄存器
0x4c	IC_CLR_TX_OVER	清除 TX_OVER 中断寄存器
0x50	IC_CLR_RD_REQ	清除 RD_REQ 中断寄存器
0x54	IC_CLR_TX_ABRT	清除 TX_ABRT 中断寄存器
0x58	IC_CLR_RX_DONE	清除 RX_DONE 中断寄存器
0x5c	IC_CLR_ACTIVITY	清除 ACTIVITY 中断寄存器
0x60	IC_CLR_STOP_DET	清除 STOP_DET 中断寄存器
0x64	IC_CLR_START_DET	清除 START_DET 中断寄存器
0x68	IC_CLR_GEN_CALL	清除 GEN_CALL 中断寄存器
0x6c	IC_ENABLE	I2C 使能寄存器
0x70	IC_STATUS	I2C 状态寄存器
0x74	IC_TXFLR	I2C 发送 FIFO 水平寄存器
0x78	IC_RXFLR	I2C 接收 FIFO 水平寄存器
0x7c	IC_SDA_HOLD	I2C SDA 保持时间寄存器
0x80	IC_TX_ABRT_SOURCE	I2C 发送中止源寄存器
0x84	IC_SLV_DATA_NACK_ONLY	生成从设备数据 NACK 寄存器
0x88	IC_DMA_CR	DMA 控制寄存器

偏移量	名称	说明
0x8c	IC_DMA_TDLR	DMA 发送数据水平寄存器
0x90	IC_DMA_RDLR	DMA 发送数据水平寄存器
0x94	IC_SDA_SETUP	I2C SDA 建立时间寄存器
0x98	IC_ACK_GENERAL_CALL	I2C 应答通用呼叫寄存器
0x9c	IC_ENABLE_STATUS	I2C 使能状态寄存器
0xa0	IC_FS_SPKLEN	I2C SS、FS 或 FM+ 峰值抑制限制
0xa8	IC_CLR_RESTART_DET	清除 RESTART_DET 中断寄存器
0xf4	IC_COMP_PARAM_1	组件参数寄存器 1
0xf8	IC_COMP_VERSION	I2C 组件版本寄存器
0xfc	IC_COMP_TYPE	I2C 组件类型寄存器

## I2C: IC\_CON 寄存器

偏移: 0x00

### 说明

I2C 控制寄存器。仅当 DW\_apb\_i2c 被禁用时（对应 IC\_ENABLE[0] 寄存器设置为 0）才能写入该寄存器。其他时间的写入操作无效。

读写权限：- 位 10 只读。- 位 11 只读，- 位 16 只读，- 位 17 只读，- 位 18 和 19 只读。

表 1056. IC\_CON 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>STOP_DET_IF_MASTER_ACTIVE</b> : 无论主设备是否处于活动状态，主设备均会触发 STOP_DET 中断	只读	0x0
9	<b>RX_FIFO_FULL_HLD_CTRL</b> : 该位控制 DW_apb_i2c 是否在 Rx FIFO 物理满至其 RX_BUFFER_DEPTH 时保持总线，详见 IC_RX_FULL_HLD_BUS_EN 参数。  复位值: 0x0。	读写	0x0
	枚举值:		
	0x0 → 禁用: 当 RX_FIFO 满时发生溢出		
	0x1 → 启用: 当 RX_FIFO 满时保持总线		
8	<b>TX_EMPTY_CTRL</b> : 该位控制 TX_EMPTY 中断的生成，详见 IC_RAW_INTR_STATUS 寄存器说明。  复位值: 0x0。	读写	0x0
	枚举值:		
	0x0 → 禁用: TX_EMPTY 中断的默认行为		
	0x1 → 启用: 受控生成 TX_EMPTY 中断		

位	描述	类型	复位
7	<p><b>STOP_DET_IFADDRESSED</b>: 在从机模式下: - 1'b1: 仅在被寻址时发出 STOP_DET 中断。- 1'b0: 无论是否被寻址, 均发出 STOP_DET 中断。复位值: 0x0</p> <p>注意: 在通用调用地址情况下, 即使从机通过生成 ACK 响应该地址, 若 STOP_DET_IF_ADDRESSED = 1'b1, 从机不会发出 STOP_DET 中断。STOP_DET 中断仅在传输地址与从机地址 (SAR) 匹配时产生。</p>	读写	0x0
	枚举值:		
	0x0 → 禁用: 从机始终发出 STOP_DET 中断		
	0x1 → 启用: 仅当地址匹配时从机发出 STOP_DET 中断		
6	<p><b>IC_SLAVE_DISABLE</b>: 此位控制 I2C 是否禁用从机功能, 即一旦施加预设信号, 该位被置位, 从机即被禁用。</p> <p>若该位被置位 (从机被禁用), DW_apb_i2c 仅作为主机运行, 不执行任何需从机身份的操作。</p> <p>注意: 软件应确保当该位写入 0 时, 第 0 位也应写入 0。</p>	读写	0x1
	枚举值:		
	0x0 → 从机启用: 从机模式已开启		
	0x1 → 从机禁用: 从机模式已关闭		
5	<p><b>IC_RESTART_EN</b>: 决定作为主机时是否允许发送 RESTART 条件。部分较旧的从机不支持处理 RESTART 条件; 然而, RESTART 条件在多种 DW_apb_i2c 操作中被使用。当禁用 RESTART 时, 主设备禁止执行以下操作: - 发送 START BYTE - 进行任何高速模式操作 - 高速模式操作 - 在组合格式模式下执行方向变更 - 使用 10 位地址执行读取操作。通过用紧接 STOP 的 RESTART 条件和随后的 START 条件替代, 分割操作被拆分为多个 DW_apb_i2c 传输。若执行上述操作, 将导致 IC_RAW_INTR_STAT 寄存器的第 6 位 (TX_ABRT) 被置位。</p> <p>复位值: ENABLED</p>	读写	0x1
	枚举值:		
	0x0 → DISABLED: 主设备重启已禁用		
	0x1 → ENABLED: 主设备重启已启用		
4	<b>IC_10BITADDR_MASTER</b> : 控制 DW_apb_i2c 作为主设备时, 传输以 7 位或 10 位地址模式启动。- 0: 7 位地址 - 1: 10 位地址	读写	0x0
	枚举值:		
	0x0 → ADDR_7BITS: 主设备 7 位地址模式		
	0x1 → ADDR_10BITS: 主控10位地址模式		

位	描述	类型	复位
3	<b>IC_10BITADDR_SLAVE:</b> 作为从机时，该位控制DW_apb_i2c响应7位或10位地址。- 0：7位寻址。DW_apb_i2c会忽略涉及10位寻址的事务；对于7位寻址，仅比较IC_SAR寄存器的低7位。- 1：10位寻址。DW_apb_i2c仅响应与IC_SAR寄存器完整10位匹配的10位寻址传输。	读写	0x0
	枚举值：		
	0x0 → ADDR_7BITS：从机7位寻址		
	0x1 → ADDR_10BITS：从机10位寻址		
2:1	<b>SPEED:</b> 这些位控制DW_apb_i2c的运行速率；仅当DW_apb_i2c工作于主控模式时，该设置才生效。 硬件防止软件编程非法值。 这些位在从模式下也必须进行适当编程，以便根据速度模式正确捕获尖峰滤波器的值。  该寄存器应仅编程为介于 1 至 IC_MAX_SPEED_MODE 范围内的值；否则，硬件将使用 IC_MAX_SPEED_MODE 的值更新该寄存器。  1：标准模式（100 kbit/s） 2：快速模式（<=400 kbit/s）或快速增强模式（<=1000 kbit/s） 3：高速模式（3.4 Mbit/s）  注意：当 IC_ULTRA_FAST_MODE=1 时，该字段不适用。	读写	0x2
	枚举值：		
	0x1 → STANDARD：标准速度操作模式		
	0x2 → FAST：快速或快速增强操作模式		
	0x3 → HIGH：高速操作模式		
0	<b>MASTER_MODE:</b> 此位用于控制是否启用 DW_apb_i2c 主控。  注意：软件应确保若该位写入“1”，则位6也应写入“1”。	读写	0x1
	枚举值：		
	0x0 → 禁用：主模式已禁用		
	0x1 → 启用：主模式已启用		

## I2C: IC\_TAR寄存器

偏移: 0x04

### 描述

I2C 目标地址寄存器

该寄存器宽度为12位，位31至12保留。仅当IC\_ENABLE[0]设置为0时，才允许写入该寄存器。

注意：若软件或应用程序知晓DW\_apb\_i2c未使用TAR地址来管理

Tx FIFO中的待处理命令，则即使Tx FIFO中有条目（IC\_STATUS[2]=0），仍可更新TAR地址。- 若DW\_apb\_i2c仅作为I2C从机启用，则无需对该寄存器进行任何写操作。

表1057. IC\_TAR 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>SPECIAL:</b> 该位指示软件是否执行设备 ID 命令、通用调用命令或起始字节命令。- 0：忽略第10位 GC_OR_START，并正常使用 IC_TAR - 1：按设备ID 或 GC_OR_START 位指定执行特殊 I2C 命令 复位值：0x0	读写	0x0
	枚举值：		
	0x0 → 禁用：禁用 GENERAL_CALL 或 START_BYTE 传输的编程		
	0x1 → 启用：启用 GENERAL_CALL 或 START_BYTE 传输的编程		
10	<b>GC_OR_START:</b> 若第11位（SPECIAL）设为1且第13位（设备 ID）设为0，则该位指示 DW_apb_i2c 是否执行通用调用或起始字节命令。- 0：通用调用地址 - 发出通用调用后，仅允许写操作。尝试发出读取命令将导致设置 IC_RA_W_INTR_STAT 寄存器的第6位（TX_ABRT）。DW_apb_i2c 将保持在General Call模式，直到SPECIAL位（第11位）被清除。- 1：START BYTE 重置值：0x0	读写	0x0
	枚举值：		
	0x0 → GENERAL_CALL：GENERAL_CALL字节传输		
	0x1 → START_BYTE：START字节传输		
9:0	<b>IC_TAR:</b> 用于任何主设备事务的目标地址。传输General Call时，这些位将被忽略。生成START BYTE时，CPU仅需写入这些位一次。  若IC_TAR与IC_SAR相同，则存在环回，但主从使用共享FIFO，无法实现完全环回。仅支持单向环回模式（单工），不支持双工模式。主设备不能向自身发送数据；只能向从设备发送数据。	读写	0x055

## I2C: IC\_SAR寄存器

偏移: 0x08

### 描述

I2C 从地址寄存器

表1058. IC\_SAR 寄存器

位	描述	类型	复位
31:10	保留。	-	-

位	描述	类型	复位
9:0	<p><b>IC_SAR:</b> 当I2C作为从设备运行时，IC_SAR保存从设备地址。对于7位寻址，仅使用IC_SAR[6:0]。</p> <p>仅当I2C接口被禁用时，即IC_ENABLE[0]寄存器设置为0，该寄存器方可写入。其他时间的写入操作无效。</p> <p>注意：默认值不得为任何保留地址，即0x00至0x07或0x78至0x7f。若将IC_SA_R或IC_TAR设置为保留值，则设备无法保证正常工作。完整保留值列表请参阅表1051。</p>	读写	0x055

## I2C: IC\_DATA\_CMD寄存器

偏移: 0x10

### 描述

I2C接收/发送数据缓冲区及命令寄存器；CPU填充TX FIFO时写入该寄存器，检索RX FIFO中字节时从该寄存器读取。

寄存器大小变化如下：

写入： - 当 IC\_EMPTYFIFO\_HOLD\_MASTER\_EN=1 时为 11 位 - 当 IC\_EMPTYFIFO\_HOLD\_MASTER\_EN=0 时为 9 位 读取：  
 - 当 IC\_FIRST\_DATA\_BYTE\_STATUS=1 时为 12 位 - 当 IC\_FIRST\_DATA\_BYTE\_STATUS=0 时为 8 位 注意：为确保 DW\_apb\_i2c 继续确认读取操作，必须为每个要接收的字节写入读取命令；  
 否则 DW\_apb\_i2c 将停止确认。

表 1059。  
IC\_DATA\_CMD  
寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<p><b>FIRST_DATA_BYTE:</b> 指示在主机接收或从机接收模式下，地址阶段后接收到的第一个数据字节。</p> <p>复位值：0x0</p> <p>注意：在 APB_DATA_WIDTH=8 的情况下，</p> <ol style="list-style-type: none"> <li>用户必须对 IC_DATA_CMD 执行两次 APB 读取操作以获取 11 位状态。</li> <li>为读取该 11 位，用户须先执行第一个数据字节的读取操作。 [7:0]（偏移量 0x10），然后执行第二次读取 [15:8]（偏移量 0x11），以确定第 11 位的状态（即先前读取的数据是否为第一个数据字节）。</li> <li>第 11 位为可选读取字段，用户可忽略第二字节的读取 [15:8] (偏移量 0x11)，若不关心 FIRST_DATA_BYTE 状态。</li> </ol>	只读	0x0
	枚举值：		
	0x0 → 非活动：接收连续数据字节		
	0x1 → 活动：接收非连续数据字节		

位	描述	类型	复位
10	<p><b>RESTART:</b> 该位控制在字节发送或接收之前是否发出重启信号。</p> <p>1 - 若 IC_RESTART_EN 为 1，则根据 CMD 的值，在数据发送/接收之前无论传输方向是否改变均发出 RESTART；若 IC_RESTART_EN 为 0，则改为先发出 STOP 信号，随后发出 START 信号。</p> <p>0 - 如果 IC_RESTART_EN 为 1，只有当传输方向相较于前一命令发生变化时才发出 RESTART；若 IC_RESTART_EN 为 0，则改为先发出 STOP 信号，随后发出 START 信号。</p> <p>复位值：0x0</p>	SC	0x0
	枚举值：		
	0x0 → 禁用：在此命令之前不发出 RESTART		
	0x1 → 启用：在此命令之前发出 RESTART		
9	<p><b>STOP:</b> 该位控制在字节发送或接收后是否发出 STOP。</p> <p>- 1 - 发送此字节后，无论 Tx FIFO 是否为空均发出 STOP。若 Tx FIFO 非空，主设备将立即尝试通过发出 START 并争用总线开始新传输。- 0 - 发送此字节后，无论 Tx FIFO 是否为空均不发出 STOP。若 Tx FIFO 非空，主设备根据 CMD 位的值继续当前传输，发送/接收数据字节。若 Tx FIFO 为空，主设备保持 SCL 线低电平，阻塞总线，直至 Tx FIFO 有新命令。复位值：0x0</p>	SC	0x0
	枚举值：		
	0x0 → 禁用：此命令后不发送STOP信号		
	0x1 → 启用：此命令后发送STOP信号		
8	<p><b>CMD:</b> 此位控制执行读操作还是写操作。该位不控制 DW_apb_i2con 作为从设备时的传输方向。它仅控制该设备作为主设备时的传输方向。</p> <p>当命令进入 TX FIFO 时，此位用于区分写命令和读命令。在从设备接收模式下，该位为“无关”状态，因为不需要写入此寄存器。在从设备发送模式下，值为“0”表示将发送 IC_DATA_CMD 中的数据。</p> <p>编程此位时应注意：发送一般呼叫命令后尝试执行读操作会触发 TX_ABRT 中断（IC_RAW_INTR_STAT 寄存器第 6 位），除非 IC_TAR 寄存器第 11 位（SPECIAL）已被清除。如果在接收到 RD_REQ 中断后将该位写为“1”，则会引发 TX_A_BRT 中断。</p> <p>复位值：0x0</p>	SC	0x0
	枚举值：		
	0x0 → 写入：主设备写命令		

位	描述	类型	复位
	0x1 → 读取：主设备读命令		
7:0	<b>DAT</b> ：此寄存器包含将在 I2C 总线上发送或接收的数据。若写入该寄存器且想执行读取操作，DW_apb_i2c 会忽略位 7:0（DAT）。但当读取此寄存器时，这些位返回 DW_apb_i2c 接口接收的数据值。  复位值：0x0	读写	0x00

## I2C: IC\_SS\_SCL\_HCNT 寄存器

偏移：0x14

### 说明

标准速率 I2C 时钟 SCL 高电平计数寄存器

表 1060。  
IC\_SS\_SCL\_HCNT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<b>IC_SS_SCL_HCNT</b> ：必须在进行任何 I2C 总线事务之前设置此寄存器，以确保正确的输入/输出时序。此寄存器设置标准速率下 SCL 时钟高电平周期计数。详情请参阅“IC_CLK 频率配置”。  仅当 I2C 接口被禁用，即 IC_ENABLE[0] 寄存器被设为 0 时，此寄存器方可写入。其他时间的写入操作无效。  最小有效值为 6；硬件防止写入小于该值的数值，若尝试写入则将自动设为 6。对于 APB_DATA_WIDTH = 8 的设计，编程顺序对于保证 DW_apb_i2c 的正确运行至关重要，必须先编程低字节。随后编程高字节。  注意：该寄存器不得被设置为大于 65525 的值，因为 DW_apb_i2c 使用 16 位计数器，当计数器达到 IC_SS_SCL_HCNT + 10 时，将标记 I2C 总线处于空闲状态。	读写	0x0028

## I2C: IC\_SS\_SCL\_LCNT 寄存器

偏移量：0x18

### 说明

标准速率 I2C 时钟 SCL 低电平计数寄存器

表 1061。  
IC\_SS\_SCL\_LCNT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15:0	<p><b>IC_SS_SCL_LCNT:</b> 此寄存器必须在任何I2C总线事务开始前设置，以确保正确的I/O时序。该寄存器用于设置标准速率下SCL时钟的低电平周期计数。详细信息，请参阅“IC_CLK 频率配置”。</p> <p>仅当I2C接口被禁用，即IC_ENABLE[0]寄存器被设为0时，此寄存器方可写入。其他时间的写入操作无效。</p> <p>最小有效值为8；硬件会阻止写入小于该值的数据，若尝试写入，则该寄存器将被设置为8。对于APB_DATA_WIDTH = 8的设计，编程顺序十分重要，以确保DW_apb_i2c的正确操作。必须先写入低字节，然后再写入高字节。</p>	读写	0x002f

## I2C: IC\_FS\_SCL\_HCNT 寄存器

偏移: 0x1c

### 说明

快速模式或快速模式 Plus I2C 时钟 SCL 高电平计数寄存器

表1062。  
IC\_FS\_SCL\_HCNT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p><b>IC_FS_SCL_HCNT:</b> 必须在任何I2C总线事务开始前设置该寄存器，以确保正确的I/O时序。该寄存器用于设置快速模式或快速模式加速下SCL时钟的高电平周期计数。该寄存器用于高速模式下发送主机代码和启动字节或通用呼叫。详细信息，请参阅“IC_CLK 频率配置”。</p> <p>当 IC_MAX_SPEED_MODE = standard 时，该寄存器将失效并变为只读，返回值为 0。仅当I2C接口被禁用时，即IC_ENABLE[0]寄存器设置为0，该寄存器方可写入。其他时间的写入操作无效。</p> <p>最小有效值为6；硬件禁止写入小于此值的数据，若尝试写入则寄存器将被设为6。对于 APB_DATA_WIDTH == 8 的设计，编程顺序极为重要，以确保 DW_apb_i2c 的正常运行。必须先编程低字节。随后编程高字节。</p>	读写	0x0006

## I2C: IC\_FS\_SCL\_LCNT 寄存器

偏移: 0x20

### 说明

快速模式或快速模式 Plus I2C 时钟 SCL 低电平计数寄存器

表 1063。  
IC\_FS\_SCL\_LCNT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15:0	<p><b>IC_FS_SCL_LCNT</b>: 必须在任何 I2C 总线事务开始之前设置此寄存器，以保证正确的 I/O 时序。该寄存器设置快速模式下 SCL 时钟低电平的计数周期。该寄存器用于高速模式下发送主机代码和启动字节或通用呼叫。详细信息，请参阅“IC_CLK 频率配置”。</p> <p>当 IC_MAX_SPEED_MODE = standard 时，该寄存器将失效并变为只读，返回值为 0。</p> <p>仅当 I2C 接口被禁用时，即 IC_ENABLE[0] 寄存器设置为 0，该寄存器方可写入。其他时间的写入操作无效。</p> <p>最小有效值为 8；硬件禁止写入小于此值的数据，若尝试写入则寄存器将被设为 8。对于 APB_DATA_WIDTH = 8 的设计，编程顺序极为重要，以确保 DW_apb_i2c 的正常运行。必须先编程低字节。然后编程高字节。如果值小于 8，则计数值被更改为 8。</p>	读写	0x000d

## I2C: IC\_INTR\_STAT 寄存器

偏移: 0x2c

### 描述

I2C 中断状态寄存器

该寄存器中的每个位均对应 IC\_INTR\_MASK 寄存器中的一个掩码位。这些位通过读取相应的中断清除寄存器来清除。这些位的未屏蔽原始状态可在 IC\_RAW\_INTR\_STAT 寄存器中获得。

表 1064  
IC\_INTR\_STAT  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<p><b>R_RESTART_DET</b>: 详见 IC_RAW_INTR_STAT 以获取 R_RESTART_DET 位的详细说明。</p> <p>复位值: 0x0</p>	只读	0x0
	枚举值:		
	0x0 → 无效: R_RESTART_DET 中断处于无效状态		
	0x1 → 有效: R_RESTART_DET 中断处于有效状态		
11	<p><b>R_GEN_CALL</b>: 详见 IC_RAW_INTR_STAT 以获取 R_GEN_CALL 位的详细说明。</p> <p>复位值: 0x0</p>	只读	0x0
	枚举值:		
	0x0 → 无效: R_GEN_CALL 中断处于无效状态		
	0x1 → 有效: R_GEN_CALL 中断处于有效状态		

位	描述	类型	复位
10	<b>R_START_DET</b> : 详见 IC_RAW_INTR_STAT 以获取 R_START_DET 位的详细说明。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_START_DET 中断处于非活动状态		
	0x1 → 活动: R_START_DET 中断处于活动状态		
9	<b>R_STOP_DET</b> : 详见 IC_RAW_INTR_STAT 以获取 R_STOP_DET 位的详细说明。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_STOP_DET 中断处于非活动状态		
	0x1 → 活动: R_STOP_DET 中断处于活动状态		
8	<b>R_ACTIVITY</b> : 详见 IC_RAW_INTR_STAT 以获取 R_ACTIVITY 位的详细说明。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_ACTIVITY 中断处于非活动状态		
	0x1 → 活动: R_ACTIVITY 中断处于活动状态		
7	<b>R_RX_DONE</b> : 详见 IC_RAW_INTR_STAT 以获取 R_RX_DONE 位的详细说明。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_RX_DONE 中断处于非活动状态		
	0x1 → 活动: R_RX_DONE 中断处于活动状态		
6	<b>R_TX_ABRT</b> : 详见 IC_RAW_INTR_STAT 以获取 R_TX_ABRT 位的详细说明。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_TX_ABRT 中断处于非活动状态		
	0x1 → 活动: R_TX_ABRT 中断处于活动状态		
5	<b>R_RD_REQ</b> : 详见 IC_RAW_INTR_STAT 以获取 R_RD_REQ 位的详细描述。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_RD_REQ 中断处于非活动状态		
	0x1 → 活动: R_RD_REQ 中断处于活动状态		

位	描述	类型	复位
4	<b>R_TX_EMPTY:</b> 详见 IC_RAW_INTR_STAT 以获取 R_TX_EMPTY 位的详细描述。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_TX_EMPTY 中断处于非活动状态		
	0x1 → 活动: R_TX_EMPTY 中断处于活动状态		
3	<b>R_TX_OVER:</b> 详见 IC_RAW_INTR_STAT 以获取 R_TX_OVER 位的详细描述。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_TX_OVER 中断处于非活动状态		
	0x1 → 活动: R_TX_OVER 中断处于活动状态		
2	<b>R_RX_FULL:</b> 详见 IC_RAW_INTR_STAT 以获取 R_RX_FULL 位的详细描述。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: R_RX_FULL 中断处于非活动状态		
	0x1 → 激活: R_RX_FULL 中断处于激活状态		
1	<b>R_RX_OVER:</b> 详见 IC_RAW_INTR_STAT 寄存器中 R_RX_OVER 位的详细描述。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 未激活: R_RX_OVER 中断处于未激活状态		
	0x1 → 激活: R_RX_OVER 中断处于激活状态		
0	<b>R_RX_UNDER:</b> 详见 IC_RAW_INTR_STAT 寄存器中 R_RX_UNDER 位的详细描述。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 未激活: RX_UNDER 中断处于未激活状态		
	0x1 → 激活: RX_UNDER 中断处于激活状态		

## I2C: IC\_INTR\_MASK 寄存器

偏移: 0x30

### 描述

I2C 中断屏蔽寄存器。

这些位用于屏蔽对应的中断状态位。该寄存器为低电平有效；值为 0 表示中断被屏蔽，值为 1 表示中断未被屏蔽。

表 1065。  
IC\_INTR\_MASK  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<b>M_RESTART_DET</b> : 该位用于屏蔽 IC_INTR_STAT 寄存器中的 R_RESTART_DET 中断。 IC_INTR_STAT 寄存器。  复位值: 0x0  枚举值:  0x0 → 启用: RESTART_DET 中断被屏蔽  0x1 → 禁用: RESTART_DET 中断未屏蔽	读写	0x0
11	<b>M_GEN_CALL</b> : 此位用于屏蔽 IC_INTR_STAT 寄存器中的 R_GEN_CALL 中断。  复位值: 0x1  枚举值:  0x0 → 启用: 屏蔽 GEN_CALL 中断  0x1 → 禁用: GEN_CALL 中断未屏蔽	读写	0x1
10	<b>M_START_DET</b> : 此位用于屏蔽 IC_INTR_STAT 寄存器中的 R_START_DET 中断。  复位值: 0x0  枚举值:  0x0 → 启用: 屏蔽 START_DET 中断  0x1 → 禁用: START_DET 中断未屏蔽	读写	0x0
9	<b>M_STOP_DET</b> : 此位用于屏蔽 IC_INTR_STAT 寄存器中的 R_STOP_DET 中断。  复位值: 0x0  枚举值:  0x0 → 启用: 屏蔽 STOP_DET 中断  0x1 → 禁用: STOP_DET 中断未屏蔽	读写	0x0
8	<b>M_ACTIVITY</b> : 此位用于屏蔽 IC_INTR_STAT 寄存器中的 R_ACTIVITY 中断。  复位值: 0x0  枚举值:  0x0 → 启用: 屏蔽 ACTIVITY 中断  0x1 → 禁用: ACTIVITY 中断未屏蔽	读写	0x0
7	<b>M_RX_DONE</b> : 该位用于屏蔽 IC_INTR_STAT 寄存器中的 R_RX_DONE 中断。 ◦  复位值: 0x1  枚举值:  0x0 → 启用: RX_DONE 中断被屏蔽	读写	0x1

位	描述	类型	复位
	0x1 → 禁用：RX_DONE中断未屏蔽		
6	<b>M_TX_ABRT</b> : 该位用于屏蔽IC_INTR_STAT寄存器中的R_TX_ABRT中断 ◦ 复位值：0x1	读写	0x1
	枚举值：		
	0x0 → 启用：TX_ABORT中断被屏蔽		
	0x1 → 禁用：TX_ABORT中断未屏蔽		
5	<b>M_RD_REQ</b> : 该位用于屏蔽IC_INTR_STAT寄存器中的R_RD_REQ中断。 ◦ 复位值：0x1	读写	0x1
	枚举值：		
	0x0 → 启用：RD_REQ中断被屏蔽		
	0x1 → 禁用：RD_REQ中断未屏蔽		
4	<b>M_TX_EMPTY</b> : 该位用于屏蔽IC_INTR_STAT寄存器中的R_TX_EMPTY中断 ◦ 复位值：0x1	读写	0x1
	枚举值：		
	0x0 → 启用：TX_EMPTY中断被屏蔽		
	0x1 → 禁用：TX_EMPTY中断未屏蔽		
3	<b>M_TX_OVER</b> : 此位用于屏蔽IC_INTR_STAT寄存器中的R_TX_OVER中断 ◦ 复位值：0x1	读写	0x1
	枚举值：		
	0x0 → 启用：TX_OVER中断被屏蔽		
	0x1 → 禁用：TX_OVER中断未屏蔽		
2	<b>M_RX_FULL</b> : 此位用于屏蔽IC_INTR_STAT寄存器中的R_RX_FULL中断 ◦ 复位值：0x1	读写	0x1
	枚举值：		
	0x0 → 启用：RX_FULL中断被屏蔽		
	0x1 → 禁用：RX_FULL中断未屏蔽		
1	<b>M_RX_OVER</b> : 此位用于屏蔽IC_INTR_STAT寄存器中的R_RX_OVER中断 ◦ 复位值：0x1	读写	0x1
	枚举值：		
	0x0 → 启用：RX_OVER中断被屏蔽		
	0x1 → 禁用：RX_OVER中断未屏蔽		

位	描述	类型	复位
0	<b>M_RX_UNDER</b> : 此位用于屏蔽IC_INTR_STAT寄存器中的R_RX_UNDER中断 ◦ 复位值: 0x1	读写	0x1
	枚举值:		
	0x0 → 启用: RX_UNDER中断被屏蔽		
	0x1 → 禁用: RX_UNDER中断未屏蔽		

## I2C: IC\_RAW\_INTR\_STAT 寄存器

偏移: 0x34

### 描述

I2C 原始中断状态寄存器

与 IC\_INTR\_STAT 寄存器不同，这些位未被屏蔽，因此始终显示 DW\_apb\_i2c 的真实状态。

表 1066。  
IC\_RAW\_INTR\_STAT  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<b>RESTART_DET</b> : 当 DW_apb_i2c 以从模式运行且从设备被寻址时，指示 I2C 接口是否发生了 RESTART 条件。仅在 IC_SLV_RESTART_DET_EN=1 时启用。  注意：根据 I2C 协议，在高速模式或 START BYTE 传输期间，RESTART 位于地址字段之前。在该情况下，RESTART 发出时从设备并非被寻址的对象，因此 DW_apb_i2c 不生成 RESTART_DET 中断。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: RESTART_DET 中断处于非活动状态		
	0x1 → 活动: RESTART_DET 中断处于活动状态		
11	<b>GEN_CALL</b> : 仅在收到并确认通用呼叫地址时置位。该状态将保持设置，直到通过禁用 DW_apb_i2c 或 CPU 读取 IC_CLR_GEN_CALL 寄存器的第 0 位时被清除。DW_apb_i2c 将接收的数据存储于 Rx 缓冲区。  复位值: 0x0	只读	0x0
	枚举值:		
	0x0 → 非活动: GEN_CALL 中断处于非激活状态		
	0x1 → 激活: GEN_CALL 中断处于激活状态		
10	<b>START_DET</b> : 指示 I2C 接口上是否发生了 START 或 RESTART 条件，无论 DW_apb_i2c 是处于主模式还是从模式。  复位值: 0x0	只读	0x0
	枚举值:		

位	描述	类型	复位
	0x0 → 非活动：START_DET 中断处于非激活状态		
	0x1 → 激活：START_DET 中断处于激活状态		
9	<p><b>STOP_DET：</b>指示 I2C 接口上是否发生了 STOP 条件，无论 DW_apb_i2c 是处于主模式还是从模式。</p> <p>在从模式下： - 若 IC_CON[7]=1'b1 (STOP_DET_IFADDRESSED) ，仅当从机被寻址时才会触发 STOP_DET 中断。注意：在一般调用地址期间，如果 STOP_DET_IF_ADDRESSED=1'b1，即使从设备通过生成ACK响应一般调用地址，也不会发出STOP_DET中断。仅当传输的地址与从设备地址 (SAR) 匹配时，才会生成STOP_DET中断。 - 如果IC_CON[7]=1'b0 (STOP_DET_IFADDRESSED) ，则无论是否被寻址，都会发出STOP_DET中断。主模式下： - 如果IC_CO N[10]=1'b1 (STOP_DET_IF_MASTER_ACTIVE) ，则仅当主设备处于活动状态时才发出STOP_DET中断。 - 如果IC_CON[10]=1'b0 (STOP_DET_IFADDRESSED) ，则无论主设备是否活动，都会发出STOP_DET中断。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		
	0x0 → 非活动：STOP_DET 中断处于非活动状态		
	0x1 → 活动：STOP_DET 中断处于活动状态		
8	<p><b>ACTIVITY：</b>该位捕捉DW_apb_i2c的活动状态，并保持置位，直至被清除。清除该位有四种方法： - 禁用DW_apb_i2c模块 - 读取IC_CLR_ACTIVITY 寄存器 - 读取IC_CLR_INTR寄存器 - 系统复位一旦此位被置位，除非采用上述任一方法清除，否则其将持续保持置位状态。即便DW_apb_i2c模块处于空闲，此位仍保持置位，表明总线上曾有活动。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		
	0x0 → 非活动状态：RAW_INTR_ACTIVITY 中断处于非活动状态		
	0x1 → 活动状态：RAW_INTR_ACTIVITY 中断处于活动状态		
7	<p><b>RX_DONE：</b>当DW_apb_i2c作为从机发送器时，若主机未确认已发送字节，此位将被置为1。该情况发生在传输的最后一个字节，表明传输已完成。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		
	0x0 → 非活动状态：RX_DONE 中断处于非活动状态		
	0x1 → 活动：RX_DONE 中断处于激活状态		

位	描述	类型	复位
6	<p><b>TX_ABRT:</b> 该位指示 DW_apb_i2c 作为 I2C 发送方时，无法完成对发送 FIFO 内容的预期操作。此情况可在 I2C 主设备或从设备模式下发生，称为“发送中止”。当此位被置为 1 时，IC_TX_ABRT_SOURCE 寄存器将指示发送中止的具体原因。</p> <p>注意：每当因 IC_TX_ABRT_SOURCE 寄存器所跟踪的任一事件导致发送中止时，DW_apb_i2c 会刷新/重置/清空 TX_FIFO 和 RX_FIFO。FIFO 将保持在此刷新状态，直到读取 IC_CLR_TX_ABRT 寄存器。读取该寄存器后，Tx FIFO 即可准备好接收来自 APB 接口的更多数据字节。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		
	0x0 → 非活动：TX_ABRT 中断处于非激活状态		
	0x1 → 活动：TX_ABRT 中断处于活动状态		
5	<p><b>RD_REQ:</b> 当 DW_apb_i2c 作为从设备，且另一 I2C 主设备试图从 DW_apb_i2c 读取数据时，该位被置为 1。DW_apb_i2c 将 I2C 总线保持在等待状态 (S CL=0)，直到该中断被处理，即从设备已被远程主设备寻址并请求传输数据。处理器必须响应此中断，随后将请求的数据写入 IC_DATA_CMD 寄存器。在处理器读取 IC_CLR_RD_REQ 寄存器后，该位被清零，置为 0。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		
	0x0 → 非活动：RD_REQ 中断处于非活动状态		
	0x1 → 活动：RD_REQ 中断处于活动状态		
4	<p><b>TX_EMPTY:</b> TX_EMPTY 中断状态的行为取决于 IC_CON 寄存器中 TX_EMPT Y_CTRL 的设置。- 当 TX_EMPTY_CTRL = 0：发送缓冲区达到或低于 IC_TX_TL 寄存器中设置的阈值时，该位置为 1。- 当 TX_EMPTY_ CTRL = 1：发送缓冲区达到或低于 IC_TX_TL 寄存器中设置的阈值，且最近弹出命令的内部移位寄存器的地址/数据传输完成时，该位置为 1。当缓冲区级别超过阈值时，该位由硬件自动清除。当 IC_ENABLE[0] 设置为 0 时，TX FIFO 会被刷新并保持复位状态。因此，TX FIFO 显示无数据，只要主机或从机状态机处于活动状态，该位即被置为 1。当无任何活动且 ic_en=0 时，该位置为 0。</p> <p>复位值：0x0。</p>	只读	0x0
	枚举值：		
	0x0 → 非激活：TX_EMPTY 中断处于非激活状态		
	0x1 → ACTIVE：TX_EMPTY 中断处于激活状态。		

位	描述	类型	复位
3	<b>TX_OVER:</b> 在发送过程中，若发送缓冲区已填满至IC_TX_BUF_FER_DEPTH且处理器试图通过写入IC_DATA_CMD寄存器发出另一条I2C命令时，该位被置位。当模块被禁用时，此位保持电平，直到主设备或从设备状态机进入空闲状态；且当ic_en为0时，该中断被清除。  复位值：0x0	只读	0x0
	枚举值：		
	0x0 → INACTIVE: TX_OVER中断处于非激活状态。		
	0x1 → ACTIVE: TX_OVER中断处于激活状态。		
2	<b>RX_FULL:</b> 当接收缓冲区达到或超过IC_RX_TL寄存器中的RX_TL阈值时置位。当缓冲区级别低于阈值时，该位由硬件自动清除。若模块被禁用（IC_ENABLE[0]=0），则RX FIFO将被清空并保持复位状态；因此，RX FIFO不会处于满状态。因此，无论后续活动如何，只要IC_ENABLE的第0位被设置为0，该位即被清除。  复位值：0x0	只读	0x0
	枚举值：		
	0x0 → 非活动：RX_FULL 中断处于非活动状态		
	0x1 → 活动：RX_FULL 中断处于活动状态		
1	<b>RX_OVER:</b> 当接收缓冲区完全填满时设置达到 IC_RX_BUFFER_DEPTH 并且从外部 I2C 设备接收了额外的一个字节。DW_apb_i2c 会对此进行确认，但在 FIFO 满载后接收的任何数据字节将丢失。如果模块被禁用（IC_ENABLE[0]=0），该位保持其电平，直到主机或从机状态机进入空闲状态，且当 ic_en 变为 0 时，该中断被清除。  注意：如果 IC_CON 寄存器的第 9 位（RX_FIFO_FULL_HLD_CTRL）被设置为高电平，则 RX_OVER 中断永远不会发生，因为 Rx FIFO 永远不会溢出。  复位值：0x0	只读	0x0
	枚举值：		
	0x0 → 非活动：RX_OVER 中断处于非活动状态		
	0x1 → 活动：RX_OVER 中断处于活动状态		
0	<b>RX_UNDER:</b> 当处理器尝试从 IC_DATA_CMD 寄存器读取空接收缓冲区时设置。如果模块被禁用（IC_ENABLE[0]=0），该位保持其电平，直到主机或从机状态机进入空闲状态，且当 ic_en 变为 0 时，该中断被清除。  复位值：0x0	只读	0x0
	枚举值：		
	0x0 → 未激活：RX_UNDER 中断处于未激活状态		
	0x1 → 激活：RX_UNDER 中断处于激活状态		

## I2C: IC\_RX\_TL 寄存器

偏移: 0x38

### 描述

I2C 接收 FIFO 阈值寄存器

表 1067. IC\_RX\_TL 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>RX_TL</b> : 接收 FIFO 门限级别。  控制触发 RX_FULL 中断 (IC_RAW_INTR_STAT 寄存器第 2 位) 的条目数或以上的门限。有效范围为 0 至 255，且硬件不允许将该值设置超过缓冲区深度。如果尝试设置更大值，实际设置值将为缓冲区的最大深度。值为 0 表示门限为 1 条目，值为 255 表示门限为 256 条目。	读写	0x00

## I2C: IC\_TX\_TL 寄存器

偏移: 0x3c

### 描述

I2C 发送 FIFO 阈值寄存器

表 1068. IC\_TX\_TL 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>TX_TL</b> : 发送 FIFO 门限级别。  控制触发 TX_EMPTY 中断 (IC_RAW_INTR_STAT 寄存器第 4 位) 的条目数或以下的门限。有效范围为 0 至 255，且不得设置为大于缓冲区深度的值。如果尝试设置更大值，实际设置值将为缓冲区的最大深度。值为 0 时，阈值设定为 0 个条目；值为 255 时，阈值设定为 255 个条目。	读写	0x00

## I2C: IC\_CLR\_INTR 寄存器

偏移量: 0x40

### 描述

清除联合及单独中断寄存器

表1069。  
IC\_CLR\_INTR寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_INTR:</b> 读取此寄存器以清除组合中断、所有单个中断以及IC_TX_ABR_T_SOURCE寄存器。此位不清除硬件可清除的中断，仅清除软件可清除的中断。有关清除IC_TX_ABRT_SOURCE的例外，请参见IC_TX_ABRT_SOURCE寄存器的第9位。  复位值：0x0	只读	0x0

## I2C: IC\_CLR\_RX\_UNDER寄存器

偏移: 0x44

### 说明

清除 RX\_UNDER 中断寄存器

表1070。  
IC\_CLR\_RX\_UNDER  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_RX_UNDER:</b> 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中RX_UNDER中断（第0位）。  复位值：0x0	只读	0x0

## I2C: IC\_CLR\_RX\_OVER寄存器

偏移: 0x48

### 说明

清除 RX\_OVER 中断寄存器

表1071。  
IC\_CLR\_RX\_OVER  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_RX_OVER:</b> 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中的RX_OVER中断（位1）。  复位值：0x0	只读	0x0

## I2C: IC\_CLR\_TX\_OVER寄存器

偏移: 0x4c

### 说明

清除 TX\_OVER 中断寄存器

表1072。  
IC\_CLR\_TX\_OVER  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_TX_OVER</b> : 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中的TX_OV ER中断（位3）。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_RD\_REQ寄存器

偏移: 0x50

### 说明

清除 RD\_REQ 中断寄存器

表1073。  
IC\_CLR\_RD\_REQ  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_RD_REQ</b> : 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中的RD_RE Q中断（位5）。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_TX\_ABRT寄存器

偏移: 0x54

### 说明

清除 TX\_ABRT 中断寄存器

表1074。  
IC\_CLR\_TX\_ABRT  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_TX_ABRT</b> : 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中的TX_AB RT中断（位6）及IC_TX_ABRT_SOURCE寄存器。此操作还可取消TX FIFO的清 空/复位状态，允许向TX FIFO执行更多写入。有关清除IC_TX_ABRT_SOURCE 的例外情况，请参阅IC_TX_ABRT_SOURCE寄存器的第9位。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_RX\_DONE寄存器

偏移: 0x58

### 描述

清除 RX\_DONE 中断寄存器

表1075。  
*IC\_CLR\_RX\_DONE*  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_RX_DONE</b> : 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中RX_DONE中断（第7位）。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_ACTIVITY 寄存器

偏移: 0x5c

### 描述

清除 ACTIVITY 中断寄存器

表1076。  
*IC\_CLR\_ACTIVITY*  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_ACTIVITY</b> : 读取此寄存器以清除ACTIVITY中断，前提是I2C已不再处于活动状态。如果I2C模块仍在总线上处于活动状态，ACTIVITY中断位将保持置位。若模块被禁用且总线无进一步活动，硬件将自动清除此中断位。读取此寄存器可获取IC_RAW_INTR_STAT寄存器中ACTIVITY中断（第8位）的状态。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_STOP\_DET 寄存器

偏移: 0x60

### 描述

清除 STOP\_DET 中断寄存器

表1077。  
*IC\_CLR\_STOP\_DET*  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_STOP_DET</b> : 读取此寄存器以清除 IC_RAW_INTR_STAT 寄存器的 STOP_DET 中断（第9位）。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_START\_DET 寄存器

偏移量: 0x64

### 说明

清除 START\_DET 中断寄存器

表 1078。  
*IC\_CLR\_START\_DET*  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_START_DET</b> : 读取此寄存器以清除 IC_RAW_INTR_STAT 寄存器的 START_DET 中断（第10位）。  复位值: 0x0	只读	0x0

## I2C: IC\_CLR\_GEN\_CALL 寄存器

偏移: 0x68

### 描述

清除 GEN\_CALL 中断寄存器

表 1079。  
*IC\_CLR\_GEN\_CALL*  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_GEN_CALL</b> : 读取此寄存器以清除 IC_RAW_INTR_STAT 寄存器的 GEN_CALL 中断（第11位）。  复位值: 0x0	只读	0x0

## I2C: IC\_ENABLE 寄存器

偏移: 0x6c

### 描述

I2C 使能寄存器

表 1080。  
*IC\_ENABLE* 寄存器

位	描述	类型	复位
31:3	保留。	-	-
2	<b>TX_CMD_BLOCK</b> : 主模式下：- 1'b1: 即使 Tx FIFO 中有数据，也阻止在 I2C 总线上发送数据。- 1'b0: 一旦 Tx FIFO 中有首个数据，数据即自动开始在 I2C 总线上传输。注意：仅当 Tx FIFO 为空 (IC_STATUS[2] == 1) 且主机处于空闲状态 (IC_STATUS[5] == 0) 时，才能设置 TX_CMD_BLOCK 位以阻止主命令执行。Tx FIFO 中的任何后续命令均不会被执行，直到 TX_CMD_BLOCK 位被清除。复位值：  IC_TX_CMD_BLOCK_DEFAULT	读写	0x0
	枚举值：		
	0x0 → NOT_BLOCKED: Tx 命令执行未被阻止		
	0x1 → BLOCKED: Tx 命令执行被阻止		

位	描述	类型	复位
1	<p><b>ABORT</b>: 设置该位时，控制器将启动传输中止操作。- 0: 未启动中止或中止已完成 - 1: 中止操作正在进行中。软件可通过设置此位在主模式下中止 I2C 传输。仅当 ENABLE 已设置时，软件方可设置此位；否则，控制器将忽略对 ABORT 位的任何写入。一旦设置，软件无法清除 ABORT 位。响应中止请求时，控制器在完成当前传输后发出 STOP 信号并刷新 Tx FIFO，随后在中止操作完成后置位 TX_ABORT 中断。ABORT 位在中止操作完成后会自动清除。</p> <p>有关中止 I2C 传输的详细说明，请参阅“中止 I2C 传输”。</p> <p>复位值: 0x0</p>	读写	0x0
	枚举值:		
	0x0 → 禁用: ABORT 操作未进行		
	0x1 → 启用: ABORT 操作正在进行中		
0	<p><b>ENABLE</b>: 用于控制是否启用 DW_apb_i2c。- 0: 禁用 DW_apb_i2c (TX 和 RX FIFO 保持清空状态) - 1: 启用 DW_apb_i2c。软件可在 DW_apb_i2c 处于活动状态时禁用该功能。但是，必须确保 DW_apb_i2c 得到正确禁用，方可保证操作安全。推荐的操作流程详见“禁用 DW_apb_i2c”。</p> <p>当 DW_apb_i2c 被禁用时，发生以下情况：发送FIFO和接收FIFO将被清空。- IC_INTR_STAT 寄存器中的状态位仍会保持有效，直至 DW_apb_i2c 进入空闲状态。如果模块正在传输，则在当前传输完成后停止传输并清除传输缓冲区的内容。如果模块正在接收，DW_apb_i2c 会在当前字节结束时停止当前传输且不予以确认。</p> <p>在异步 pclk 和 ic_clk 的系统中，当 IC_CLK_TYPE 参数设置为异步 (1) 时，启用或禁用 DW_apb_i2c 会存在两个 ic_clk 时钟的延迟。关于如何禁用 DW_apb_i2c 的详细说明，请参阅“禁用 DW_apb_i2c”。</p> <p>复位值: 0x0</p>	读写	0x0
	枚举值:		
	0x0 → 禁用: I2C 被禁用		
	0x1 → 启用: I2C 被启用		

## I2C: IC\_STATUS 寄存器

偏移量: 0x70

### 描述

I2C 状态寄存器

本寄存器为只读，指示当前传输状态及 FIFO 状态。状态寄存器可在任何时间读取。该寄存器中的各位均不触发中断请求。

当通过在 IC\_ENABLE 寄存器的第0位写入0以禁用 I2C 时：- 第1位和第2位被置为1 - 第3位和第10位被置为0 当主机或从机状态机进入空闲状态且 ic\_en=0 时：- 第5位和第6位被置为0

表1081。  
IC\_STATUS 寄存器

位	描述	类型	复位
31:7	保留。	-	-
6	<b>SLV_ACTIVITY:</b> 从机 FSM 活动状态。当从机有限状态机 (FSM) 不处于 IDLE 状态时, 该位被置位。- 0: 从机 FSM 处于 IDLE 状态, DW_apb_i2c 的从机部分不活跃 -1: 从机 FSM 不处于 IDLE 状态, DW_apb_i2c 的从机部分活跃 复位值: 0x0  枚举值: 0x0 → IDLE: 从机空闲 0x1 → ACTIVE: 从机非空闲	只读	0x0
5	<b>MST_ACTIVITY:</b> 主机 FSM 活动状态。当主机有限状态机 (FSM) 不处于 IDLE 状态时, 该位被置位。- 0: 主机 FSM 处于 IDLE 状态, DW_apb_i2c 的主机部分不活跃 -1: 主机 FSM 不处于 IDLE 状态, DW_apb_i2c 的主机部分活跃 注: IC_STATUS[0]——即ACTIVITY位——是SLV_ACTIVITY位和MST_ACTIVITY位的逻辑或。  复位值: 0x0  枚举值: 0x0 → 空闲: 主控处于空闲状态 0x1 → 活动: 主控不处于空闲状态	只读	0x0
4	<b>RFF:</b> 接收FIFO已满。当接收FIFO完全满时, 该位被置位; 当接收FIFO含有一个或多个空位时, 该位被清除。- 0: 接收FIFO未满 -1: 接收FIFO已满 复位值: 0x0  枚举值: 0x0 → 未满: 接收FIFO未满 0x1 → 已满: 接收FIFO已满	只读	0x0
3	<b>RFNE:</b> 接收FIFO非空。当接收FIFO包含一个或多个条目时, 该位被置位; 当接收FIFO为空时, 该位被清除。- 0: 接收FIFO为空 -1: 接收FIFO非空 复位值: 0x0  枚举值: 0x0 → 空: 接收FIFO为空 0x1 → NOT_EMPTY: 接收FIFO非空	只读	0x0
2	<b>TFE:</b> 发送FIFO完全清空。当发送FIFO完全清空时, 该位被置位; 当其包含一个或多个有效条目时, 该位被清除。该位字段不触发中断请求。- 0: 发送FIFO非空 -1: 发送FIFO为空 复位值: 0x1  枚举值: 0x0 → NON_EMPTY: 发送FIFO非空 0x1 → EMPTY: 发送FIFO为空	只读	0x1
1	<b>TFNF:</b> 发送FIFO未满。当发送FIFO含有一个或多个空闲位置时, 该位被置位; 当FIFO满时, 该位被清除。- 0: 发送FIFO已满 -1: 发送FIFO未满 复位值: 0x1  枚举值:	只读	0x1

位	描述	类型	复位
	0x0 → FULL：发送FIFO已满		
	0x1 → NOT_FULL：发送FIFO未满		
0	<b>ACTIVITY</b> ：I2C活动状态。复位值：0x0	只读	0x0
	枚举值：		
	0x0 → 非活动：I2C处于空闲状态		
	0x1 → 活动：I2C处于活动状态		

## I2C：IC\_TXFLR寄存器

偏移: 0x74

### 描述

I2C发送FIFO级别寄存器，该寄存器包含发送FIFO缓冲区中有效数据条目的数量。

每当出现以下情况时，该寄存器将被清除：- I2C被禁用 - 发生发送中止，即IC\_RAW\_INTR\_STAT寄存器中的TX\_ABRT位被置位 - 从设备批量发送模式被中止。数据被写入发送FIFO时，寄存器递增；数据从发送FIFO读取时，寄存器递减。

表1082。IC\_TXFLR  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4:0	<b>TXFLR</b> ：发送FIFO级别，包含发送FIFO中有效数据条目的数量。  复位值：0x0	只读	0x00

## I2C：IC\_RXFLR寄存器

偏移：0x78

### 描述

I2C接收FIFO级别寄存器，该寄存器包含接收FIFO缓冲区中有效数据条目的数量。每当出现以下情况时，该寄存器将被清除：- I2C被禁用 - 由于IC\_TX\_ABRT\_SOURCE中跟踪的任一事件导致发送中止。数据被写入接收FIFO时，寄存器递增；数据从接收FIFO读取时，寄存器递减。

表1083。IC\_RXFLR  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4:0	<b>RXFLR</b> ：接收FIFO级别。包含接收FIFO中有效数据条目的数量。  复位值：0x0	只读	0x00

## I2C：IC\_SDA\_HOLD寄存器

偏移：0x7c

### 说明

I2C SDA 保持时间寄存器

此寄存器的位[15:0]用于控制从机和主机模式下传输期间SDA的保持时间  
(在SCL从高电平变为低电平之后)。

此寄存器的位[23:16]用于在接收端主机或从机模式下，当SCL为高电平时，延长SDA的转换(如有)。

无论是在主机还是从机模式。

仅当IC\_ENABLE[0]=0时，写入此寄存器才会成功。

该寄存器中的数值以ic\_clk周期为单位。写入IC\_SDA\_TX\_HOLD的值必须大于各模式的最小时长（主机模式为一个周期，从机模式为七个周期），方能生效。

传输过程中编程的SDA保持时间(IC\_SDA\_TX\_HOLD)在任何时刻均不得超过SCL低电平的持续时间。因此，编程值不得大于N\_SCL\_LOW-2，其中N\_SCL\_LOW是以ic\_clk周期为单位的SCL低电平持续时间。

表1084。  
IC\_SDA\_HOLD  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>IC_SDA_RX_HOLD:</b> 当DW_apb_i2c作为接收方时，设置所需的SDA保持时间，单位为ic_clk周期。  复位值：IC_DEFAULT_SDA_HOLD[23:16]。	读写	0x00
15:0	<b>IC_SDA_TX_HOLD:</b> 当DW_apb_i2c作为发送方时，设置所需的SDA保持时间，单位为ic_clk周期。  复位值：IC_DEFAULT_SDA_HOLD[15:0]。	读写	0x0001

## I2C: IC\_TX\_ABRT\_SOURCE寄存器

偏移: 0x80

### 描述

I2C 发送中止源寄存器

该寄存器包含32位，用于指示TX\_ABRT位的来源。除第9位外，每当读取IC\_CLR\_TX\_ABRT寄存器或IC\_CLR\_INTR寄存器时，均清除此寄存器。要清除第9位，必须先修正ABRT\_SBYTE\_NORSTRT的源；必须启用RESTART (IC\_CON[5]=1)，清除SPECIAL位 (IC\_TAR[11])，或清除GC\_OR\_START位 (IC\_TAR[10])。

一旦ABRT\_SBYTE\_NORSTRT的源头被修复，便可采用与清除此寄存器中其他位相同的方式清除此位。若在尝试清除此位前，ABRT\_SBYTE\_NORSTRT的源头未被修复，则第9位将清零一个时钟周期后重新置位。

表1085。  
IC\_TX\_ABRT\_SOURCE  
寄存器

位	描述	类型	复位
31:23	<b>TX_FLUSH_CNT:</b> 该字段指示因TX_ABRT中断而被清除的Tx FIFO数据命令数量。每当I2C被禁用时，该字段即被清零。  复位值：0x0  DW_apb_i2c的角色：主机发送器或从机发送器	只读	0x000
22:17	保留。	-	-
16	<b>ABRT_USER_ABRT:</b> 此位仅适用于主机模式。主机已检测到传输中止 (IC_ENABLE[1])。  复位值：0x0  DW_apb_i2c的角色：主机发送器	只读	0x0
	枚举值：		

位	描述	类型	复位
	0x0 → ABRT_USER_ABRT_VOID：主机未检测到传输中止情况		
	0x1 → ABRT_USER_ABRT_GENERATED：主控检测到传输中止		
15	<b>ABRT_SLVRD_INTX:</b> 1: 当处理器响应远程主控的从机模式数据传输请求，且用户在IC_DATA_CMD寄存器的CMD（第8位）写入1时。  复位值: 0x0  DW_apb_i2c的角色: 从设备-发送者	只读	0x0
	枚举值:		
	0x0 → ABRT_SLVRD_INTX_VOID: 从机尝试在读取模式向远程主控发送数据—此场景不存在		
	0x1 → ABRT_SLVRD_INTX_GENERATED: 从机尝试在读取模式向远程主控发送数据		
14	<b>ABRT_SLV_ARBLOST:</b> 该字段指示从设备在向远程主控传输数据时丢失总线控制权。IC_TX_ABRT_SOURCE[12]同时被置位。注意: 尽管从机从未“拥有”总线，传输过程仍可能出现总线故障。此为故障保护检查。例如，在SCL信号由低电平跃变为高电平的数据传输过程中，若数据总线上的内容不是预期传输的数据，则DW_apb_i2c将不再拥有该总线。  复位值: 0x0  DW_apb_i2c的角色: 从设备-发送者	只读	0x0
	枚举值:		
	0x0 → ABRT_SLV_ARBLOST_VOID: 从设备因失去与远程主控设备的仲裁而中止—未出现该场景		
	0x1 → ABRT_SLV_ARBLOST_GENERATED: 从设备因失去与远程主控设备的仲裁而中止		
13	<b>ABRT_SLVFLUSH_TXFIFO:</b> 该字段指示从设备已接收到读命令且TX FIFO中存在数据，故从设备触发TX_ABRT中断以刷新TX FIFO内的旧数据。  复位值: 0x0  DW_apb_i2c的角色: 从设备-发送者	只读	0x0
	枚举值:		
	0x0 → ABRT_SLVFLUSH_TXFIFO_VOID: 从设备在接收到读命令时刷新TX FIFO内现有数据—未出现该场景		
	0x1 → ABRT_SLVFLUSH_TXFIFO_GENERATED: 从设备在接收到读命令时刷新TX FIFO内现有数据		

位	描述	类型	复位
12	<p><b>ARB_LOST:</b> 该字段指示主设备失去仲裁，或当 IC_TX_ABRT_SOURCE[14] 同时被设置时，表明从设备发送器失去仲裁。</p> <p>复位值：0x0</p> <p>DW_apb_i2c的角色：主机发送器或从机发送器</p>	只读	0x0
	枚举值：		
	0x0 → ABRT_LOST_VOID：主设备或从设备发送器未发生失去仲裁的情形		
	0x1 → ABRT_LOST_GENERATED：主设备或从设备发送器失去仲裁		
11	<p><b>ABRT_MASTER_DIS:</b> 该字段指示用户尝试在主模式禁用状态下启动主操作。</p> <p>复位值：0x0</p> <p>DW_apb_i2c 的角色：主设备发送器或主设备接收器</p>	只读	0x0
	枚举值：		
	0x0 → ABRT_MASTER_DIS_VOID：用户在主模式禁用时启动主操作的情况不存在		
	0x1 → ABRT_MASTER_DIS_GENERATED：用户在主模式禁用时启动主操作		
10	<p><b>ABRT_10B_RD_NORSTRT:</b> 该字段指示重启功能被禁用 (IC_RESTART_EN 位 (IC_CON[5]) =0) ，且主设备在 10 位地址模式下发送读取命令。</p> <p>复位值：0x0</p> <p>DW_apb_i2c 的角色：主设备-接收器</p>	只读	0x0
	枚举值：		
	0x0 → ABRT_10B_RD_VOID：当 RESTART 禁用时，主设备未尝试在 10 位地址模式下读取		
	0x1 → ABRT_10B_RD_GENERATED：当 RESTART 禁用时，主设备尝试在 10 位地址模式下读取		
9	<p><b>ABRT_SBYTE_NORSTRT:</b> 要清除第 9 位，必须首先解决 ABRT_SBYTE_NORSTRT 的根本原因；必须启用 restart (IC_CON[5] =1) ，必须清除 SPECIAL 位 (IC_TAR[11]) ，或者必须清除 GC_OR_START 位 (IC_TAR[10]) 。一旦解决了 ABRT_SBYTE_NORSTRT 的根本原因，该位即可像本寄存器中的其他位一样清除。如果在尝试清除此位之前未解决 ABRT_SBYTE_NORSTRT 的根本原因，第 9 位清除一个周期后重新断言。当此字段设置为1时，重启被禁用 (IC_RESTART_EN 位 (IC_CON[5])=0) ，且用户试图发送 START 字节。</p> <p>复位值：0x0</p> <p>DW_apb_i2c 的角色：主设备</p>	只读	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → ABRT_SBYTE_NORSTRT_VOID：重启被禁用时用户试图发送 START 字节——此情况不存在		
	0x1 → ABRT_SBYTE_NORSTRT_GENERATED：重启被禁用时用户试图发送 START 字节		
8	<b>ABRT_HS_NORSTRT</b> ：该字段表明重启被禁用 (IC_RESTART_EN 位 (IC_CON[5])=0) , 且用户试图使用主设备在高速模式下传输数据。  复位值：0x0  DW_apb_i2c 的角色：主设备发送器或主设备接收器	只读	0x0
	枚举值：		
	0x0 → ABRT_HS_NORSTRT_VOID：重启被禁用时用户试图将主设备切换至高速模式——此情况不存在		
	0x1 → ABRT_HS_NORSTRT_GENERATED：用户尝试在禁用重启功能时将主机切换至高速模式		
7	<b>ABRT_SBYTE_ACKDET</b> ：该字段指示主机已发送启动字节且启动字节被应答（错误行为）  复位值：0x0  DW_apb_i2c 的角色：主设备	只读	0x0
	枚举值：		
	0x0 → ABRT_SBYTE_ACKDET_VOID：未检测到启动字节应答——此场景不存在		
	0x1 → ABRT_SBYTE_ACKDET_GENERATED：检测到启动字节应答		
6	<b>ABRT_HS_ACKDET</b> ：该字段指示主机处于高速模式且高速主机代码被应答（错误行为）  复位值：0x0  DW_apb_i2c 的角色：主设备	只读	0x0
	枚举值：		
	0x0 → ABRT_HS_ACK_VOID：高速模式下高速主机代码应答——此场景不存在		
	0x1 → ABRT_HS_ACK_GENERATED：高速模式下高速主机代码应答		
5	<b>ABRT_GCALL_READ</b> ：该字段指示DW_apb_i2c在主机模式下发送了广播调用，但用户将广播调用后的字节设置为从总线上读取 (IC_DATA_CMD[9] 被设置为1)  复位值：0x0  DW_apb_i2c的角色：主机发送器	只读	0x0
	枚举值：		

位	描述	类型	复位
	0x0 → ABRT_GCALL_READ_VOID： GCALL后接从总线读取—该场景不存在		
	0x1 → ABRT_GCALL_READ_GENERATED： GCALL后接从总线读取		
4	<b>ABRT_GCALL_NOACK：</b> 此字段指示在主模式下，DW_apb_i2c发送了General Call，但总线上没有任何从设备响应该General Call。  复位值：0x0  DW_apb_i2c的角色：主机发送器	只读	0x0
	枚举值：		
	0x0 → ABRT_GCALL_NOACK_VOID：无从设备响应GCALL—该场景不存在		
	0x1 → ABRT_GCALL_NOACK_GENERATED：无从设备响应GCALL		
3	<b>ABRT_TXDATA_NOACK：</b> 此字段仅适用于主模式。当主设备已收到地址确认，但在发送后续数据字节时未收到远端从设备的确认，该位被置位。  复位值：0x0  DW_apb_i2c的角色：主机发送器	只读	0x0
	枚举值：		
	0x0 → ABRT_TXDATA_NOACK_VOID：所有被寻址的从设备均未确认所发送数据—该场景不存在		
	0x1 → ABRT_TXDATA_NOACK_GENERATED：发送的数据未被指定的从机确认		
2	<b>ABRT_10ADDR2_NOACK：</b> 该字段指示主设备处于10位地址模式，且10位地址的第二个地址字节未被任何从机确认。  复位值：0x0  DW_apb_i2c 的角色：主设备发送器或主设备接收器	只读	0x0
	枚举值：		
	0x0 → INACTIVE：未生成该中止信号		
	0x1 → ACTIVE：10位地址的第2字节未被任何从机确认		
1	<b>ABRT_10ADDR1_NOACK：</b> 该字段指示主设备处于10位地址模式，且10位地址的第一个地址字节未被任何从机确认。  复位值：0x0  DW_apb_i2c 的角色：主设备发送器或主设备接收器	只读	0x0
	枚举值：		
	0x0 → INACTIVE：未生成该中止信号		

位	描述	类型	复位
	0x1 → ACTIVE：10位地址的第1字节未被任何从机确认		
0	<b>ABRT_7B_ADDR_NOACK：</b> 该字段指示主设备处于7位地址模式，且发送的地址未被任何从机确认。  复位值：0x0  DW_apb_i2c 的角色：主设备发送器或主设备接收器	只读	0x0
	枚举值：		
	0x0 → INACTIVE：未生成该中止信号		
	0x1 → ACTIVE：因7位地址未被确认而生成该中止信号。		

## I2C: IC\_SLV\_DATA\_NACK\_ONLY 寄存器

偏移量：0x84

### 描述

生成从设备数据 NACK 寄存器

当 DW\_apb\_i2c 作为从机接收时，该寄存器用于生成传输数据部分的 NACK。

仅当 IC\_SLV\_DATA\_NACK\_ONLY 参数设置为 1 时，才存在该寄存器。当该参数被禁用时，该寄存器不存在，对该寄存器地址的写操作无任何效果。

仅当满足以下两个条件时，才允许对该寄存器进行写操作：- DW\_apb\_i2c 被禁用 (IC\_ENABLE[0] = 0) - 从机部分处于非活动状态 (IC\_STATUS[6] = 0) 注：IC\_STATUS[6] 是内部 slv\_activity 信号的寄存器读回位；用户应在写入 ic\_slv\_data\_nack\_only 位之前对该位进行轮询检查。

表 1086。  
IC\_SLV\_DATA\_NACK\_ONLY 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>NACK：</b> 生成NACK。此NACK生成仅在DW_apb_i2c作为从机接收器时发生。 如果此寄存器设置为1，则仅能在接收数据字节后生成NACK；因此，数据传输将中止，接收到的数据不会写入接收缓冲区。  当寄存器设置为0时，将根据正常标准生成NACK或ACK。- 1：数据字节接收后生成NACK - 0：按正常方式生成NACK/ACK 重置值：0x0	读写	0x0
	枚举值：		
	0x0 → 禁用：从机接收器按正常方式生成NACK		
	0x1 → 启用：从机接收器仅在接收数据时生成NACK		

## I2C: IC\_DMA\_CR 寄存器

偏移量：0x88

### 描述

DMA控制寄存器

该寄存器用于启用DMA控制器接口的操作功能。传输和接收各有独立控制位。

无论 IC\_ENABLE 状态如何，均可进行编程。

表1087。  
IC\_DMA\_CR 寄存器

位	描述	类型	复位
31:2	保留。	-	-

位	描述	类型	复位
1	<b>TDMAE:</b> 发送 DMA 使能位。该位用于启用或禁用发送 FIFO DMA 通道。复位值: 0x0。	读写	0x0
	枚举值:		
	0x0 → 禁用: 发送 FIFO DMA 通道已禁用		
	0x1 → 启用: 发送 FIFO DMA 通道已启用		
0	<b>RDMAE:</b> 接收 DMA 使能位。该位用于启用或禁用接收 FIFO DMA 通道。复位值: 0x0。	读写	0x0
	枚举值:		
	0x0 → 禁用: 接收 FIFO DMA 通道已禁用		
	0x1 → 启用: 接收 FIFO DMA 通道已启用		

## I2C: IC\_DMA\_TDLR 寄存器

偏移: 0x8c

### 描述

DMA 发送数据水平寄存器

表1088。  
IC\_DMA\_TDLR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	<b>DMATDL:</b> 发送数据水位值。该字段控制发送逻辑发起 DMA 请求的阈值。其数值等同于水位线水平；即当发送 FIFO 中的有效数据条目数小于或等于此字段值且 TDMAE=1 时，dma_tx_req 信号将被生成。  复位值: 0x0	读写	0x0

## I2C: IC\_DMA\_RDLR 寄存器

偏移: 0x90

### 描述

I2C 接收数据级别寄存器

表 1089。  
IC\_DMA\_RDLR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	<b>DMARDL:</b> 接收数据级别。该位字段控制接收逻辑发出 DMA 请求的阈值。水位线等级 = DMARDL + 1；即，当接收 FIFO 中有效数据条目数大于或等于该字段值加 1 且 RDMAE = 1 时，产生 dma_rx_req。例如，当 DMARDL 为 0 时，只要接收 FIFO 中存在 1 个或以上的数据条目，即断言 dma_rx_req。  复位值: 0x0	读写	0x0

## I2C: IC\_SDA\_SETUP 寄存器

偏移: 0x94

**说明**

I2C SDA 设置寄存器

该寄存器控制在 DW\_apb\_i2c 作为从机发送器响应读请求时，SCL 上升沿相对于 SDA 变化所引入的延迟时间（单位为 ic\_clk 时钟周期数）。相关的I2C要求为tSU:DAT（注4），详见I2C总线规范。该寄存器须编程为等于或大于2的值。

仅当IC\_ENABLE[0] = 0时，写入该寄存器操作方能成功。

注：设定时间长度计算公式为[(IC\_SDA\_SETUP - 1) \* (ic\_clk\_period)]，因此若用户需求为10个ic\_clk周期的设定时间，则应编程为11。 IC\_SDA\_SETUP寄存器仅用于DW\_apb\_i2c作为从设备发送器时。

表1090。  
IC\_SDA\_SETUP  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>SDA_SETUP：</b> SDA设定。建议当所需延迟为1000ns时，且ic_clk频率为10 MHz，IC_SDA_SETUP应编程为11。 IC_SDA_SETUP的最小编程值为2。	读写	0x64

## I2C: IC\_ACK\_GENERAL\_CALL 寄存器

偏移: 0x98

**描述**

I2C 应答通用呼叫寄存器

该寄存器用于控制 DW\_apb\_i2c 在接收到 I2C 通用呼叫地址时，是否以 ACK 或 NACK 响应。

该寄存器仅在 DW\_apb\_i2c 处于从模式时适用。

表 1091。  
IC\_ACK\_GENERAL\_CA  
LL 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>ACK_GEN_CALL：</b> ACK 通用呼叫。设置为 1 时，DW_apb_i2c 在接收到通用呼叫时通过断言 ic_data_oe 以 ACK 响应。否则，DW_apb_i2c 通过否定 ic_data_oe 以 NACK 响应。	读写	0x1
	枚举值：		
	0x0 → 禁用：对通用呼叫生成 NACK		
	0x1 → 启用：对通用呼叫生成 ACK		

## I2C: IC\_ENABLE\_STATUS 寄存器

偏移量: 0x9c

**描述**

I2C 使能状态寄存器

该寄存器用于报告在 IC\_ENABLE[0] 寄存器由 1 变为 0 时，DW\_apb\_i2c 硬件的状态；即，当DW\_apb\_i2c被禁用时。

如果IC\_ENABLE[0]已设置为1，则位2:1被强制清零，且位0被强制置1。

若IC\_ENABLE[0]已设置为0，则仅当位0被读取为“0”时，位2:1才有效。

注意：当IC\_ENABLE[0]设置为0时，位0被读取为0存在延迟，这因禁用DW\_apb\_i2c取决于I2C总线活动情况。

表1092。  
IC\_ENABLE\_STATUS  
寄存器

位	描述	类型	复位
31:3	保留。	-	-
2	<p><b>SLV_RX_DATA_LOST</b>: 从机接收数据丢失。该位指示当IC_ENABLE的位0由1变为0时，是否中止了至少接收过一个I2C传输数据字节的从机接收操作。当读取值为1时，DW_apb_i2c 被视为已主动参与一场被中止的I2C传输（地址匹配），且已进入I2C传输的数据阶段，尽管数据字节已被响应为NACK。</p> <p>注意：如果远程I2C主设备在DW_apb_i2c有机会对传输发出NACK之前通过STOP条件终止传输，且IC_ENABLE[0]已设置为0，则此位亦被置为1。</p> <p>当读取值为0时，DW_apb_i2c被视为已禁用，且未主动参与从机接收传输的数据阶段。</p> <p>注意：当IC_EN（第0位）读取为0时，CPU可安全读取此位。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		
	0x0 → 非活动：从机接收数据未丢失		
	0x1 → 活动：从机接收数据已丢失		
1	<p><b>SLV_DISABLED_WHILE_BUSY</b>: 从机在忙碌（发送、接收）期间被禁用。该位指示由于将IC_ENABLE寄存器第0位由1清零，潜在或活跃的从机操作被中止。当CPU向IC_ENABLE寄存器写入0且符合以下条件时，该位被置位：</p> <p>(a) DW_apb_i2c 正在从远程主机接收从属发送操作的地址字节；</p> <p>或者，</p> <p>(b) 来自远程主机的从属接收操作的地址字节和数据字节。</p> <p>读取值为1时，DW_apb_i2c 被视为在任何 I2C 传输阶段强制发送了 NACK，无论 I2C 地址是否与 DW_apb_i2c （IC_SAR 寄存器）中设置的从设备地址匹配，亦或传输是否在 IC_ENABLE 设置为0前完成但尚未生效。</p> <p>注意：若远程 I2C 主机在 DW_apb_i2c 有机会发送 NACK 之前以 STOP 条件终止传输，且 IC_ENABLE[0] 已设置为0，则该位亦将被置为1。</p> <p>读取值为0时，DW_apb_i2c 被视为在主机活动期间或 I2C 总线空闲时已被禁用。</p> <p>注意：当IC_EN（第0位）读取为0时，CPU可安全读取此位。</p> <p>复位值：0x0</p>	只读	0x0
	枚举值：		

位	描述	类型	复位
	0x0 → 非活动：当从机空闲时被禁用		
	0x1 → 活动：当从机激活时被禁用		
0	<b>IC_EN</b> : ic_en 状态。该位始终反映输出端口的驱动值 ic_en。- 读取为1时，DW_apb_i2c 被视为启用状态。- 读取为0时，DW_apb_i2c 被视为完全非活动状态。注意：CPU 可随时安全读取此位。当此位读取为0时，CPU 可安全读取 SLV_RX_DATA_LOST (第2位) 和 SLV_DISABLED_WHILE_BUSY (第1位)。  复位值：0x0	只读	0x0
	枚举值：		
	0x0 → 禁用：I2C 被禁用		
	0x1 → 启用：I2C 被启用		

## I2C: IC\_FS\_SPKLEN 寄存器

偏移：0xa0

### 描述

I2C SS、FS 或 FM+ 峰值抑制限制

该寄存器用于存储最长尖峰的持续时间，单位为 ic\_clk 周期，该尖峰由尖峰抑制逻辑在组件处于 SS、FS 或 FM+ 模式下运行时过滤。相关的I2C要求为tSP（见表4），详见I2C总线规范。该寄存器必须编程为不小于1的值。

表1093。  
IC\_FS\_SPKLEN  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>IC_FS_SPKLEN</b> : 此寄存器必须在任何I2C总线事务开始之前设置，以确保操作稳定。此寄存器设定以ic_clk周期计量的SCL或SDA线上的最长尖峰持续时间，峰值抑制逻辑将对此尖峰进行过滤。仅当I2C接口被禁用，即IC_ENABLE[0]寄存器被设为0时，此寄存器方可写入。其他时间写入无效。最小有效值为1；硬件禁止写入小于此值的数据，若尝试写入，将设为1。详情请参阅“峰值抑制”。	读写	0x07

## I2C: IC\_CLR\_RESTART\_DET寄存器

偏移: 0xa8

### 描述

清除 RESTART\_DET 中断寄存器

表1094。  
IC\_CLR\_RESTART\_DET  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLR_RESTART_DET</b> : 读取此寄存器以清除IC_RAW_INTR_STAT寄存器中RESTART_DET中断（第12位）标志。  复位值: 0x0	只读	0x0

## I2C: IC\_COMP\_PARAM\_1 寄存器

偏移量: 0xf4

### 说明

组件参数寄存器 1

注意: 该寄存器未实现, 因此读取值为0。若已实现, 该寄存器将为只读常量, 包含组件参数设置信息的编码内容。以下字段展示了这些参数的设置

表 1095  
IC\_COMP\_PARAM\_1  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>TX_BUFFER_DEPTH</b> : TX 缓冲区深度 = 16	只读	0x00
15:8	<b>RX_BUFFER_DEPTH</b> : RX 缓冲区深度 = 16	只读	0x00
7	<b>ADD_ENCODED_PARAMS</b> : 编码参数不可见	只读	0x0
6	<b>HAS_DMA</b> : 启用 DMA 握手机制	只读	0x0
5	<b>INTR_IO</b> : 组合中断输出	只读	0x0
4	<b>HC_COUNT_VALUES</b> : 各模式的可编程计数值	只读	0x0
3:2	<b>MAX_SPEED_MODE</b> : 最大速度模式 = 快速模式	只读	0x0
1:0	<b>APB_DATA_WIDTH</b> : APB 数据总线宽度为 32 位	只读	0x0

## I2C: IC\_COMP\_VERSION 寄存器

偏移量: 0xf8

### 说明

I2C 组件版本寄存器

表 1096  
IC\_COMP\_VERSION  
寄存器

位	描述	类型	复位
31:0	<b>IC_COMP_VERSION</b>	只读	0x3230312a

## I2C: IC\_COMP\_TYPE 寄存器

偏移: 0xfc

### 描述

I2C 组件类型寄存器

表 1097。  
*IC\_COMP\_TYPE*  
寄存器

位	描述	类型	复位
31:0	<b>IC_COMP_TYPE</b> : Designware 组件类型编号 = 0x44_57_01_40。此指定的唯一十六进制值为恒定值，由两个 ASCII 字母 'DW' 后跟一个16位无符号数构成。	只读	0x44570140

## 12.3. SPI

Arm 文档

摘自 ARM PrimeCell 同步串行端口 (PL022) 技术参考手册。经授权使用。

RP2350 配备两个相同的 SPI 控制器，均基于 Arm PrimeCell 同步串行端口 (SSP) (PL022) (版本 r1p4)。该接口不同于第 12.14 节中介绍的 QSPI 存储器接口。

每个控制器支持以下功能：

- 主模式或从模式
  - Motorola SPI 兼容接口
  - Texas Instruments 同步串行接口
  - National Semiconductor Microwire 接口
- 8位置的TX和RX FIFO
- 生成中断以服务FIFO或指示错误状态
- 支持 DMA 驱动
- 可编程时钟速率
- 可编程数据宽度4-16位

每个控制器可连接至银行0 GPIO功能表 (表646, 第9.4节) 中定义的若干GPIO引脚。

GPIO功能表中的条目，如“SPI0 TX”，指定该GPIO引脚上的SPI实例及对应的SPI信号。表中信号描述如下：

### SCK

串行时钟。连接到以下章节中描述的SPI外设时钟信号 *SSPCLKOUT* 和 *SSPCLKIN*。在从设备模式下，该引脚为输入；在主设备模式下为输出。

### TX

串行数据输出。连接到以下章节中描述的SPI外设信号 *SSPTXD* (数据输出) 和 *nSSPOE* (输出使能)。该信号始终为数据输出，与总线角色无关。SPI 外设根据芯片选择信号状态控制三态输出。

### RX

串行数据输入。连接至 SPI 外设的 *SSPRXD* 数据输入，如后续章节所述。该信号始终作为数据输入，与总线角色无关。

### CSn

低电平有效芯片选择信号。连接至 SPI 外设信号 *SSPFSSOUT* 和 *SSPFSSIN*，如后续章节所述。在从设备模式下，该引脚为输入；在主设备模式下为输出。

SPI 使用 *clk\_peri* 作为 SPI 定时的参考时钟，以下章节中称为 *SSPCLK*。 *clk\_sys* 用作总线时钟，以下章节中称为 *PCLK* (另见图 33)。

### 12.3.1. 与 RP2040 的变更

`SSPTXD`数据输出（连接至 GPIO 功能表中列出的 SPI0 TX 和 SPI1 TX 引脚）的输出使能由 SPI 外设信号 `nSSPOE` 控制。外设在从属模式下被取消选择时自动进入三态输出。即使多个从设备共享数据线，也无需软件控制输出使能。

### 12.3.2. 概述

PrimeCell SSP 是用于与具备 Motorola SPI、National Semiconductor Microwire 或 Texas Instruments 同步串行接口的外围设备进行同步串行通信的主从接口。

PrimeCell SSP 对从外围设备接收的数据执行串行转并行转换。CPU 通过 AMBA APB 接口访问数据、控制及状态信息。传输和接收路径均采用内部 FIFO 缓冲器，允许在传输和接收模式下分别独立存储最多八个 16 位值。串行数据通过 `SSPTXD` 发送，并通过 `SSPRXD` 接收。

PrimeCell SSP 配备可编程位速率时钟分频器和预分频器，用于从输入时钟 `SSPCLK` 生成串行输出时钟 `SSPCLKOUT`。位速率支持达到 2MHz 及更高，具体取决于 `SSPCLK` 频率的选择，最大位速率由外围设备决定。

您可以使用控制寄存器 `SSPCR0` 和 `SSPCR1` 来配置 PrimeCell SSP 的操作模式、帧格式及其大小。

会产生以下可单独屏蔽的中断：

- `SSPTXINTR` 请求对发送缓冲区进行服务
- `SSPRXINTR` 请求对接收缓冲区进行服务
- `SSPRORINTR` 指示接收 FIFO 产生溢出情况
- `SSPRTINTR` 指示在接收 FIFO 中有数据时超时时间已到。

若任一单独中断被断言且未被屏蔽，则会断言单一组合中断。该中断连接至 RP2350 处理器的中断控制器。

除了上述中断外，还提供一组 DMA 信号以实现与 DMA 控制器的接口。

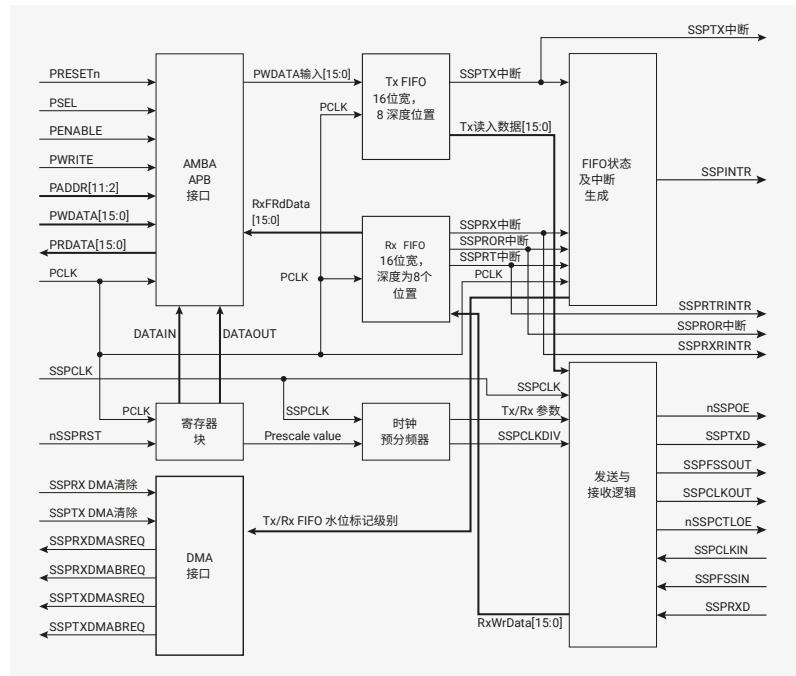
根据所选操作模式，`SSPFSSOUT` 输出的功能如下：

- 作为德州仪器同步串行帧格式的高电平有效帧同步输出
- 作为 SPI 和 Microwire 的低电平有效从机选择信号。

### 12.3.3. 功能描述

图91。PrimeCell SSP框图。

为清晰起见，不显示测试逻辑。



### 12.3.3.1. AMBA APB接口

AMBA APB接口为访问状态和控制寄存器以及发送与接收FIFO存储器生成读写译码。

### 12.3.3.2. 寄存器块

寄存器块用于通过AMBA APB接口存储已写入或待读取的数据。

### 12.3.3.3. 时钟分频器

当配置为主设备时，内部分频器由两个串联的自由运行可重载计数器组成，提供串行输出时钟 **SSPCLKOUT**。

您可以通过 **SSPCPSR** 寄存器对时钟分频器进行编程，以两个为步长将 **SSPCLK** 分频为2至254的倍数。由于未使用 **SSPCPSR** 寄存器的最低有效位，无法实现奇数分频；这确保生成的时钟具有对称的高低电平占空比。详见**SSPCPSR**。

分频器的输出通过编程**SSPCRO**控制寄存器，再次按1-256的因子分频，以生成最终的主输出时钟 **SSPCLKOUT**。

#### **i 注意**

图91中，PCLK和SSPCLK时钟输入分别连接至RP2350的系统级时钟网络 **clk\_sys**和**clk\_peri**。默认情况下，**clk\_peri**直接连接至系统时钟。但若系统时钟动态变化，可将其断开以保持SPI频率恒定。有关RP2350时钟架构的整体概述，请参见图33。

### 12.3.3.4. 发送FIFO

通用发送（TX）FIFO为16位宽、深度为8个位置的内存缓冲区。通过AMBA接口写入的CPU数据

APB接口的写入数据存储于缓冲区中，直到由发送逻辑读取。

配置为主机或从机时，平行数据写入发送FIFO，随后进行串行转换，通过 **SSPTXD**引脚分别传输至连接的从机或主机。

### 12.3.3.5. 接收FIFO

通用接收（RX）FIFO是一个16位宽、8个位置深的存储缓冲区。从串行接口接收的数据会存储在缓冲区中，直到CPU通过AMBA APB接口将其读取。

配置为主设备或从设备时，通过 **SSPRXD**引脚接收的串行数据，在并行装载到所连接的从设备或主设备接收FIFO之前，会被寄存。

### 12.3.3.6. 发送和接收逻辑

配置为主设备时，所连接从设备的时钟来源于经过上述分频器操作处理的 **SSPCLK**分频信号。主设备发送逻辑依次从发送FIFO读取数据，并进行并行转串行转换。随后，同步于 **SSPCLKOUT**的串行数据流及帧控制信号通过 **SSPTXD**引脚输出至所连接的从设备。主设备接收逻辑对输入的同步 **SSPRXD**串行数据流执行串行转并行转换，提取并存储数据至接收FIFO，供后续通过APB接口读取。

当配置为从属设备时，**SSPCLKIN**时钟由连接的主设备提供，用于定时传输和接收序列。在主时钟控制下，从属发送逻辑依次执行：

1. 从其发送FIFO中读取一个值。
2. 执行并行转串行转换。
3. 通过从属设备的 **SSPTXD**引脚输出串行数据流和帧控制信号。

从属接收逻辑对输入的 **SSPRXD**数据流执行串行转并行转换，提取并存储值至其接收FIFO，以供后续通过APB接口读取。

### 12.3.3.7. 中断生成逻辑

PrimeCell SSP产生四个独立的可屏蔽、有效高电平中断。组合中断输出由各单独中断请求的逻辑或产生。

发送和接收动态数据流中断，分别为 **SSPTXINTR**和 **SSPRXINTR**，与状态中断分离，以便根据FIFO触发级别进行数据的读写操作。

### 12.3.3.8. DMA接口

PrimeCell SSP提供了一个连接DMA控制器的接口，详见第12.3.4.16节。

### 12.3.3.9. 寄存器及逻辑同步

PrimeCell SSP支持时钟的异步和同步操作，分别为 **PCLK**和 **SSPCLK**。

同步寄存器及握手机制已实现，并始终处于激活状态。控制信号的同步在数据流的两个方向均进行，即：

- 从 **PCLK**域到 **SSPCLK**域
- 从 **SSPCLK**域到 **PCLK**域。

## 12.3.4. 操作

### 12.3.4.1. 接口复位

PrimeCell SSP由全局复位信号 `PRESETn`以及模块专用复位信号 `nSSPRST`复位。设备复位控制器异步断言 `nSSPRST`，并在与 `SSPC_LK`同步时取消断言。

### 12.3.4.2. 配置SSP

复位后，PrimeCell SSP逻辑被禁用，必须在此状态下进行配置。必须编程控制寄存器 `SSPCR0`和 `SSPCR1`，以配置外设为主设备或从设备，并使其按照以下协议之一工作：

- 摩托罗拉 SPI
- 德州仪器 SSI
- 国家半导体

位速率由外部 `SSPCLK`派生，需编程时钟预分频寄存器 `SSPCPSR`。

### 12.3.4.3. 启用PrimeCell SSP操作

您可以在PrimeCell SSP禁用时，通过写入最多八个16位值来预置发送FIFO，或允许发送FIFO服务请求中断CPU。一旦启用，数据的传输或接收即通过发送引脚 `SSPTXD`和接收引脚 `SSPRXD`开始。

### 12.3.4.4. 时钟比率

对 `PCLK`与 `SSPCLK`的频率比存在约束。`SSPCLK`的频率必须小于或等于 `PCLK`的频率。这确保来自 `SSPCLK`域到 `PCLK`域的控制信号在一个帧周期内得到同步：

$$F_{SSPCLK} \leq F_{PCLK}$$

在从机工作模式下，来自外部主机的 `SSPCLKIN`信号经过双重同步处理后再延迟，以用于边缘检测。检测 `SSPCLKIN`上的边缘需要三个 `SSPCLK`周期。`SSPTXD`相对于主机采样线路的 `SSPCLKIN`下降沿具有较短的建立时间。

相对于 `SSPCLKIN`的 `SSPRXD`建立和保持时间必须更加保守，以确保在 `SSPMS`内的实际采样时信号值正确。为确保设备正确运行，`SSPCLK`的频率必须至少是最大预期 `SSPCLKIN`频率的12倍。

所选的 `SSPCLK`频率必须满足所需的位时钟速率范围。在从属模式下，最小 `SSPCLK`频率与 `SSPCLKOUT`最大频率的比率为12，主模式下该比率为2。

例如，在RP2350的最大 `SSPCLK`频率（`clk_peri`）150MHz下，主模式的最大峰值比特率为70.5Mb/s。此值通过将 `SSPCPSR`寄存器设置为2，以及将 `SSPCR0`寄存器中的`SCR[7:0]`字段设置为0实现。

在从属模式下，最大 `SSPCLK`频率同为150MHz，可实现峰值比特率为 $150 \div 12 = 12.5$ Mb/s。将 `SSPCPSR`寄存器设置为12，并将 `SSPCR0`寄存器中的`SCR[7:0]`字段设置为0即可。类似地，`SSPCLK`最大频率与 `SSPCLKOUT`最小频率的比率为 $254 \times 256$ 。

`SSPCLK`的最小频率受以下不等式约束，且必须同时满足：

$$F_{SSPCLK}(min) \geq 2 \times F_{SSPCLKOUT}(max), \text{ 适用于主模式}$$

$F_{SSPCLK}(min) \geq 12 \times F_{SSPCLKIN}(max)$ , 适用于从模式。

SSPCLK的最大频率受以下不等式约束, 且必须同时满足:

$F_{SSPCLK}(max) \leq 254 \times 256 \times F_{SSPCLKOUT}(min)$ , 适用于主模式

$F_{SSPCLK}(max) \leq 254 \times 256 \times F_{SSPCLKIN}(min)$ , 适用于从模式。

#### 12.3.4.5. 编程 SSPCR0 控制寄存器

SSPCR0寄存器用于:

- 编程串行时钟速率
- 选择三种协议中的一种
- 选择数据字长 (如适用)

串行时钟速率 (SCR) 值结合 SSPCSR时钟预分频因子值 CPSDVS, 用于从外部 SSPCLK派生PrimeCell SSP的发送与接收比特率。

帧格式通过FRF位设置, 数据字长通过 DSS位配置。

位相和极性, 仅适用于Motorola SPI格式, 通过 SPH和 SPO位进行配置。

#### 12.3.4.6. SSPCR1控制寄存器的编程

SSPCR1寄存器用于:

- 选择主模式或从模式
- 启用环回测试功能
- 启用PrimeCell SSP外设。

要将PrimeCell SSP配置为主模式, 请将SSPCR1寄存器中的主从选择位MS清零。该值为复位时的默认值。

将SSPCR1寄存器中的MS位置1, 可将PrimeCell SSP配置为从模式。配置为从模式时, 使用SSPCR1寄存器中从模式SSPTXD输出禁用位 (S0D) 启用或禁用PrimeCell SSP的SSPTXD信号。该功能适用于某些多从设备环境中, 主设备可能进行并行广播的场合。

要启用PrimeCell SSP, 请将同步串行端口使能位 (SSE) 置位为1。

##### 12.3.4.6.1. 比特率生成

串行比特率通过对输入时钟 SSPCLK进行分频得出。时钟首先通过一个范围为2至254的偶数预分频值CPSDVS分频, 该值在 SSPCSR中设置。时钟随后由1至256范围内的值再次分频, 即 $1 + SCR$ , SCR的值由 SSPCR0中设置。

以下公式定义了输出信号比特时钟 SSPCLKOUT 的频率:

$$F_{SSPCLKOUT} = \frac{F_{SSPCLK}}{CPSDVS \times (1 + SCR)}$$

例如, 当 SSPCLK为125MHz且 CPSDVS= 2时, SSPCLKOUT的频率范围为244kHz至62.5MHz。

#### 12.3.4.7. 帧格式

每个数据帧长为4至16位, 具体取决于所编程的数据大小, 且从最高有效位 (MSB) 开始传输。您可以选择以下基本帧类型:

- 德州仪器同步串行
- 摩托罗拉 SPI
- 国家半导体Microwire。

对于所有格式，串行时钟 **SSPCLKOUT** 在 PrimeCell SSP 空闲时保持非活动状态，仅在数据的主动传输或接收期间以设定频率跳变。利用 **SSPFSSOUT** 的空闲状态提供接收超时指示，该指示发生于接收 FIFO 在超时期限后仍含有数据时。

对于 Motorola SPI 和国家半导体 Microwire 帧格式，串行帧信号 **SSPFSSOUT** 引脚为有效低电平，且在整个帧传输期间被拉低。

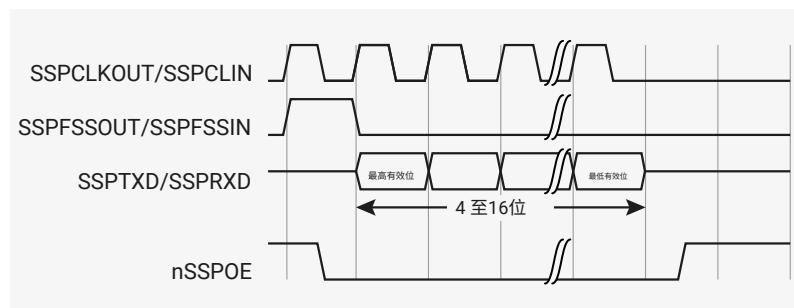
对于德州仪器同步串行帧格式，**SSPFSSOUT** 引脚在每个帧传输前的上升沿开始脉冲一个串行时钟周期。对于此帧格式，PrimeCell SSP 及片外从设备均在 **SSPCLKOUT** 上升沿驱动输出数据，并在下降沿锁存来自对方设备的数据。

与其他两种帧格式的全双工传输不同，National Semiconductor Microwire 格式采用一种特殊的主从通信技术，运行于半双工模式。在此模式下，帧开始时，会向片外从设备发送一个 8 位的控制消息。在该传输过程中，SSS 不接收任何输入数据。消息发送完成后，片外从设备会进行解码，并在 8 位控制消息最后一位发送后等待一个串行时钟周期，然后返回所请求的数据。返回的数据长度可为 4 至 16 位，使总帧长度处于 13 至 25 位之间。

#### 12.3.4.8. Texas Instruments 同步串行帧格式

图92显示了Texas Instruments单帧同步串行帧格式。

图92. 德州仪器同步串行帧格式，单次传输

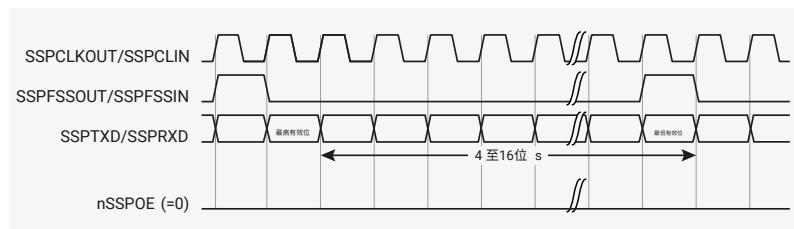


在此模式下，**SSPCLKOUT** 和 **SSPFSSOUT** 被强制拉低，且 PrimeCell SSP 空闲时，发送数据线 **SSPTXD** 处于三态状态。当发送 FIFO 底部条目包含数据时，**SSPFSSOUT** 会被拉高一个 **SSPCLKOUT** 周期的脉冲。传输的数据也从发送 FIFO 传送至发送逻辑的串行移位寄存器。在下一次 **SSPCLKOUT** 的上升沿，4 位至 16 位数据帧的最高有效位（MSB）通过 **SSPTXD** 引脚移出。类似地，接收数据的最高有效位由片外串行从设备加载到 **SSPRXD** 引脚。

随后，PrimeCell SSP 和片外串行从设备均在每个 **SSPCLKOUT** 的下降沿将各数据位时钟进入其串行移位寄存器。接收的数据在锁存最低有效位（LSB）后，于 **PCLK** 的首个上升沿从串行移位寄存器传输至接收 FIFO。

图93显示了德州仪器在背靠背帧传输时的同步串行帧格式。

图93. 德州仪器同步串行帧格式，连续传输



### 12.3.4.9. 摩托罗拉SPI帧格式

摩托罗拉SPI接口为四线接口，其中 **SSPFSSOUT**信号用作从设备选择信号。摩托罗拉SPI格式的主要特点是可以通过 **SSPSCRO** 控制寄存器中的**SP0**和**SPH**位，配置**SSPCLKOUT**信号的无效状态及相位。

#### 12.3.4.9.1. **SP0**, 时钟极性

当 **SP0**时钟极性控制位为低电平时，**SSPCLKOUT**引脚保持稳定的低电平；当 **SP0**时钟极性控制位为高电平，且无数据传输时，**SSPCLKOUT**引脚保持稳定的高电平。

#### 12.3.4.9.2. **SPH**, 时钟相位

该 **SPH**控制位选择捕获数据的时钟边缘，并允许其状态发生变化。它对首比特的传输影响最大，决定是否允许在第一个数据捕获边缘之前发生时钟跳变。

当 **SPH**相位控制位为低电平时，数据在第一个时钟边缘跳变处被捕获。

当 **SPH**时钟相位控制位为高电平时，数据在第二个时钟边缘跳变处被捕获。

### 12.3.4.10. **SPO=0, SPH=0**的Motorola SPI格式

图94和图95展示了**SPO=0, SPH=0**的Motorola SPI帧格式的连续传输信号序列。图94展示了**SPO=0, SPH=0**的Motorola SPI帧格式的单次传输信号序列。

图94。Motorola SPI帧格式，单次传输，**SP0=0**且**SPH=0**

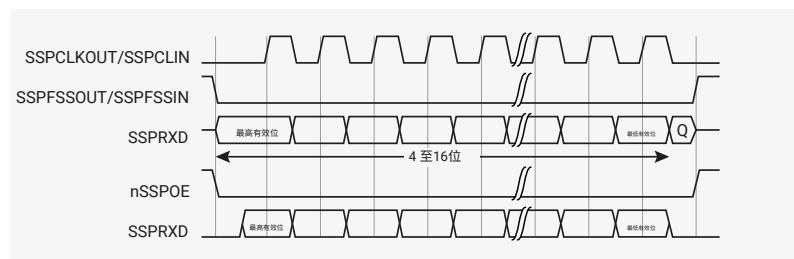
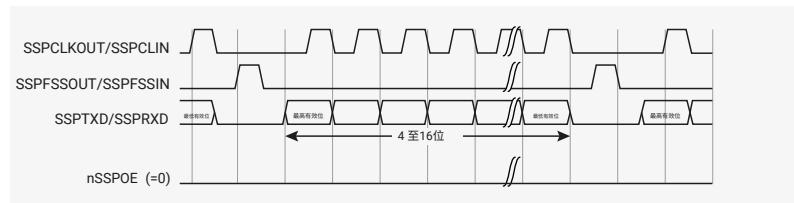


图95显示了摩托罗拉SPI帧格式下连续传输信号序列，**SP0=0, SPH=0**。

图95。Motorola SPI帧格式，单次传输，**SP0=0**且**SPH=0**



在该配置下，空闲期间：

- **SSPCLKOUT**信号被强制拉低
- **SSPFSSOUT**信号被强制拉高
- 传输数据线**SSPTXD**被任意强制拉低
- **nSSPOE**端口使能信号被强制拉高（该信号在RP2350中未连接至端口）
- 当PrimeCell SSP配置为主设备时，**nSSPCTL0E**线被拉低，启用**SSPCLKOUT**端口，低电平有效使能
- 当PrimeCell SSP配置为从设备时，**nSSPCTL0E**线被拉高，禁用**SSPCLKOUT**端口，低电平有效使能

若PrimeCell SSP处于使能状态且传输FIFO中有有效数据，传输开始由SSPFSSOUT主设备信号拉低指示，进而使从设备数据能够加载至主设备的SSPRXD输入线上。nSSPOE线被拉低，启用主控端SSPTXD输出引脚。

半个 SSPCLKOUT周期后，有效的主控数据传输到 SSPTXD引脚。主从数据均已设定后，主控时钟引脚 SSPCLKOUT在再过半个 SSP CLKOUT周期后拉高。

数据现于 SSPCLKOUT信号的上升沿捕获，并在下降沿传播。

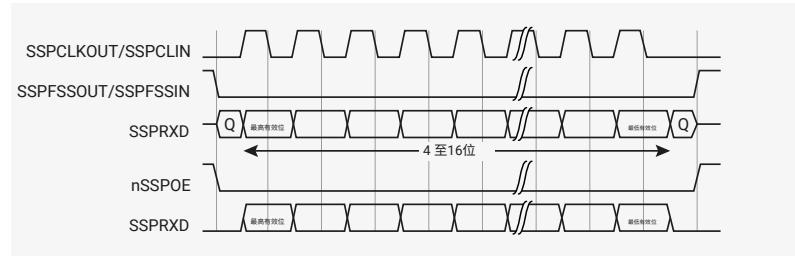
对于单字传输，所有数据位传输完成后，最后一位被捕获一个 SSPCLKOUT周期后， SSPFSSOUT线恢复至空闲高电平状态。

但在连续回连传输中， SSPFSSOUT信号在每个数据字传输间保持高电平脉冲。这是因为当 SPH位逻辑值为零时，从属选择引脚会冻结其串行外设寄存器中的数据，不允许修改。因此，主设备必须在每次数据传输之间将从设备的 SSPFSSIN引脚拉高，以启用串行外设数据写入。连续传输完成后，SSPFSSOUT引脚将在最后一位被捕获后一个SSPCLKOUT周期内恢复至空闲状态。

#### 12.3.4.11. SPO=0, SPH=1 的 Motorola SPI 格式

图96显示了 SPO=0、SPH=1 的 Motorola SPI 格式传输信号序列，涵盖单次传输和连续传输。

图96。 SPO=0 且  
SPH=1 的 Motorola S  
PI 帧格式，单次  
传输与连续传输



在该配置下，空闲期间：

- SSPCLKOUT信号被强制拉低
- SSPFSSOUT信号被强制置为高电平
- 传输数据线SSPTXD被任意强制拉低
- nSSPOE引脚使能信号被强制置为高电平（RP2350 中未连接至引脚）
- 当PrimeCell SSP配置为主设备时，nSSPC<sub>TLOE</sub>线被拉低，启用SSPCLKOUT端口，低电平有效使能
- 当PrimeCell SSP配置为从设备时，nSSPC<sub>TLOE</sub>线被拉高，禁用SSPCLKOUT端口，低电平有效使能

若启用 PrimeCell SSP 且发送 FIFO 中存在有效数据，传输开始即由将 SSPFSSOUT主信号拉低表示。nSSPOE线被拉低，启用主设备的SSPTXD输出引脚。再经过半个SSPCLKOUT周期后，主、从双方的有效数据被同时激活并传输至各自线路。同时，SSPCLKOUT通过上升沿触发被使能。

数据随后在 SSPCLKOUT信号的下降沿被捕获，并在上升沿传输。

在单字传输的情况下，当所有位传输完成后，SSPFSSOUT线会在最后一位被捕获后的一个 SSPCLKOUT周期内恢复至其空闲的高电平状态。对于连续背靠背传输，SSPFSSOUT引脚在连续数据字之间保持低电平，终止方式与单字传输相同。

### 12.3.4.12. Motorola SPI格式，SPO=1，SPH=0

图97和图98显示了SPO=1，SPH=0配置下的Motorola SPI格式的单次及连续传输信号序列。

图97显示了SPO=1，SPH=0配置下的Motorola SPI格式的单次传输信号序列。

图97。Motorola SPI帧格式，单次传输，SPO=1且SPH=0

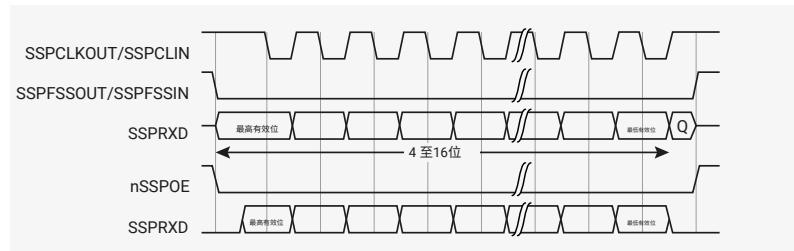
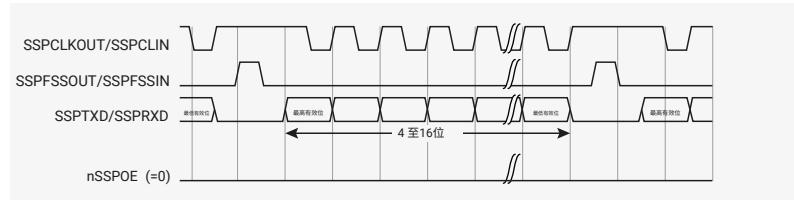


图98显示了Motorola SPI格式连续传输的信号序列，SPO=1，SPH=0。

#### **i 注意**

在图97中，Q为未定义信号。

图98。Motorola SPI帧格式，连续传输，SPO=1且SPH=0



在该配置下，空闲期间：

- SSPCLKOUT信号被强制置高
- SSPFSSOUT信号被强制拉高
- 传输数据线SSPTXD被任意强制拉低
- nSSPOE引脚使能信号被强制置为高电平（RP2350 中未连接至引脚）
- 当PrimeCell SSP配置为主设备时，nSSPC<sub>TLOE</sub>线被拉低，启用SSPCLKOUT端口，低电平有效使能
- 当PrimeCell SSP配置为从设备时，nSSPC<sub>TLOE</sub>线被拉高，禁用SSPCLKOUT端口，低电平有效使能

若启用PrimeCell SSP且发送FIFO中有有效数据，传输启动由SSPFSSOUT主信号拉低表示，该操作使从设备数据立即传送至主设备的SSPRXD线。nSSPOE线被拉低，启用主控端SSPTXD输出引脚。

半周期后，有效的主设备数据被传输至SSPTXD线。既然主从设备数据均已设定，SSPCLKOUT主时钟引脚在额外经过半个SSPCLKOUT周期后变为低电平。这意味着数据在SSPCLKOUT信号的下降沿被采集，并在上升沿传输。

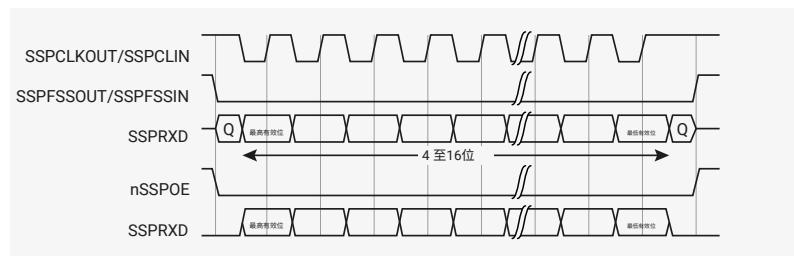
在单字传输的情况下，所有数据位传输完成后，SSPFSSOUT线路将在最后一位采集后的一个SSPCLKOUT周期内恢复为空闲高电平状态。

然而，在连续背靠背传输时，SSPFSSOUT信号必须在每个数据字传输之间产生高电平脉冲。这是因为当SPH位逻辑值为零时，从属选择引脚会冻结其串行外设寄存器中的数据，不允许修改。因此，主设备必须在每次数据传输之间将从设备的SSPFSSIN引脚拉高，以启用串行外设数据写入。连续传输结束后，SSPFSSOUT引脚将在最后一位采集后的一个SSPCLKOUT周期恢复至空闲状态。

### 12.3.4.13. Motorola SPI 格式 (SPO=1, SPH=1)

图 99 展示了 SPO=1、SPH=1 条件下 Motorola SPI 格式的传输信号序列，涵盖单次及连续传输。

图 99。Motorola  
SPI 帧格式，SPO=1  
且 SPH=1，支持  
单次和连续传输。



**i 注意**

在图 99 中，信号 Q 未定义。

在该配置下，空闲期间：

- SSPCLKOUT 信号被强制置高
- SSPFSSOUT 信号被强制拉高
- 传输数据线 SSPTXD 被任意强制拉低
- nSSPOE 引脚使能信号被强制置为高电平 (RP2350 中未连接至引脚)
- 当 PrimeCell SSP 配置为主设备时，nSSPC<sub>TLOE</sub> 线被拉低，启用 SSPCLKOUT 端口，低电平有效使能
- 当 PrimeCell SSP 配置为从设备时，nSSPC<sub>TLOE</sub> 线被拉高，禁用 SSPCLKOUT 端口，低电平有效的使能信号。

若启用 PrimeCell SSP，且传输 FIFO 中存在有效数据，则传输开始时由 SSPFSSOUT 主控信号拉低。同时，nSSPOE 线路拉低，启用主控 SSPTXD 输出端。

再经过半个 SSPCLKOUT 周期后，主控与从控数据同时驱动至各自传输线。与此同时，SSPCLKOUT 由下降沿触发并被使能。数据随后在 SSPCLKOUT 信号的上升沿被采样，并在下降沿被传播。

所有位传输完成后，对于单词传输，最后一位采样完成后的 SSPCLKOUT 周期内，SSPFSSOUT 线路恢复至空闲高电平。

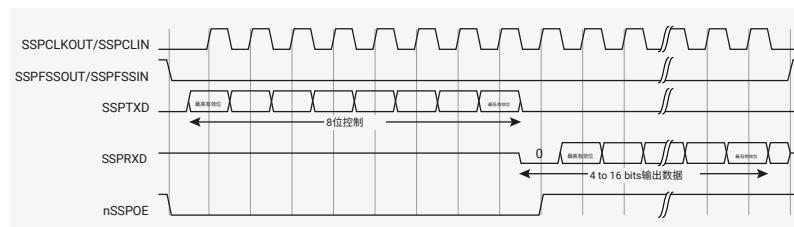
对于连续背靠背传输，SSPFSSOUT 引脚维持低电平有效状态，直至最后一个字的最后一一位被捕获，随后如前述部分所述恢复到空闲状态。

对于连续背靠背传输，SSPFSSOUT 引脚在连续数据字之间保持低电平，终止方式与单字传输相同。

### 12.3.4.14. 国家半导体Microwire帧格式

图100展示了单帧国家半导体Microwire帧格式。图101展示了连续帧传输时的相同帧格式。

图100。Microwire  
帧格式，单次  
传输



Microwire格式与SPI格式极为相似，不同之处在于其传输为半双工而非全双工，采用主从消息传递机制。每次串行传输始于由PrimeCell SSP向片外从设备发送的8位控制字。在此传输过程中，PrimeCell SSP未接收任何输入数据。消息发送后，外部芯片从属设备对其进行解码，并在发送完8位控制消息的最后一一位后等待一个串行时钟周期，随后响应所需数据。返回的数据长度为4至16位，使总帧长度在13至25位之间。

在该配置下，空闲期间：

- **SSPCLKOUT** 被强制拉低
- **SSPFSSOUT** 被强制拉高
- 发送数据线 **SSPTXD**被任意拉低
- **nSSPOE**引脚使能信号被强制置为高电平（RP2350 中未连接至引脚）

通过向发送FIFO写入控制字节触发传输。**SSPFSSOUT**的下降沿使发送FIFO底端的值传输至发送逻辑的串行移位寄存器，8位控制帧的最高有效位被移出至 **SSPTXD**引脚。**SSPFSSOUT**在帧传输期间保持低电平。**SSPRXD**引脚在此次传输中保持三态状态。

片外串行从设备在每个 **SSPCLKOUT**的上升沿将每个控制位锁存至其串行移位寄存器。在从设备锁存最后一位后，控制字节将在一个时钟等待状态内解码，从设备随后通过向PrimeCell SSP发送数据进行响应。每个位于**SSPCLKOUT**的下降沿驱动至 **SSPRXD**线上。PrimeCell SSP则于**SSPCLKOUT**的上升沿锁存每个位。在帧结束时，对于单次传输，**SSPFSSOUT**信号在接收串行移位寄存器锁存最后一位后一个时钟周期被拉高，从而将数据传输到接收FIFO。

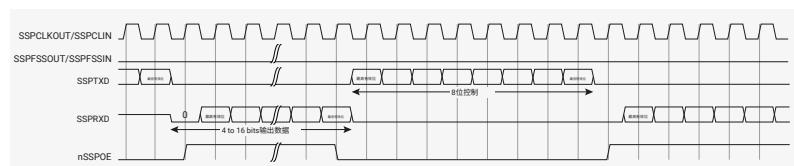
### **i 注意**

片外从设备可在接收移位器锁存LSB之后的 **SSPCLKOUT**下降沿，或 **SSPFSSOUT**引脚变高时，将接收线置于三态。

对于连续传输，数据传输的开始和结束方式与单次传输相同。然而，**SSPFSSOUT**线持续被拉低且被保持，数据传输紧密连续进行。下一帧的控制字节紧接在当前帧接收数据的最低有效位（LSB）之后。每个接收值均在**SSPCLKOUT**的下降沿从接收移位器传输，此时帧的LSB已锁存至PrimeCell SSP。

图101展示了National Semiconductor Microwire在连续帧传输时的帧格式。

图101。Microwire  
帧格式，连  
续传输



在Microwire模式下，PrimeCell SSP从机于SSPFSSIN信号拉低后，在SSPCLKIN的上升沿采样接收数据的首位。驱动自由运行SSPCLKIN的主机必须确保SSPFSSIN信号相较于SSPCLKIN的上升沿具备足够的建立时间和保持时间裕度。

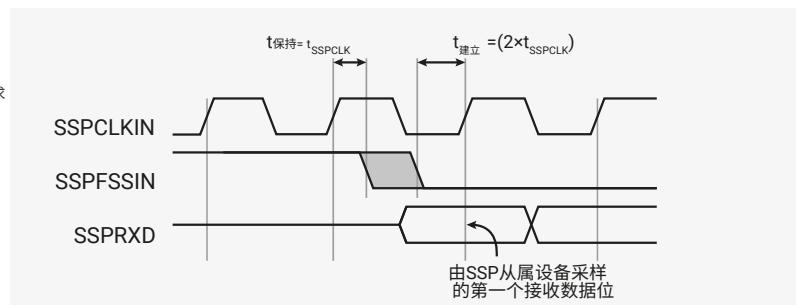
图102显示了这些建立时间和保持时间的要求。

关于PrimeCell SSP从属设备采样第一个接收数据位的 **SSPCLKIN**上升沿，**SSPFSSIN**必须满足至少为PrimeCell SSP工作 **SSPCLK**周期两倍的建立时间。

关于该边沿之前的 **SSPCLKIN**上升沿，**SSPFSSIN**必须满足至少一个 **SSPCLK**周期的保持时间。

图102。Microwire

帧格式  
**SSPFSSIN**输入的建  
立时间和保持时间要求



#### 12.3.4.15. 主设备与从设备配置示例

图103、图104和图105展示了在PrimeCell SSP（PL022）外设被配置为主设备或从设备时，如何连接至其他同步串行外设。

**i 注意**

SSP（PL022）不支持系统中主从动态切换。每个实例均配置并连接为主设备或从设备。

图103展示了PrimeCell SSP（PL022）两次实例化，分别作为一个主设备和一个从设备。主设备可以通过其 SSPTXD线向从设备广播。作为响应，从设备将其 nSSPOE信号置高，使其SSPTXD数据输出到主设备的 SSPRXD线上。

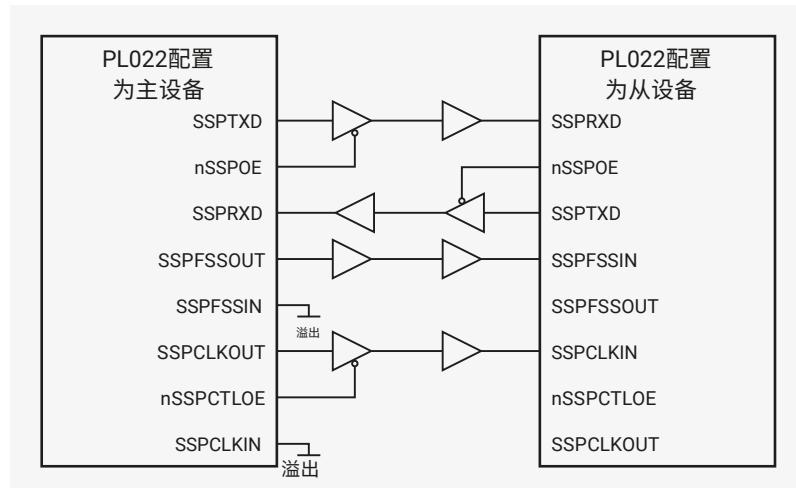
图103。PrimeCell  
SSP主设备连接至  
PL022从设备

图104展示了PrimeCell SSP（PL022）作为主设备时如何与摩托罗拉SPI从设备接口。SPI从设备选择（SS）信号被永久拉低，配置为从设备。类似上述操作，主设备可通过PrimeCell SSP主控的 SSPTXD线向从设备广播。作为响应，从设备将其 SPI MISO端口驱动至主设备的 SSPRXD线路。

图104。连接至SPI  
从设备的PrimeCell SSP主设备

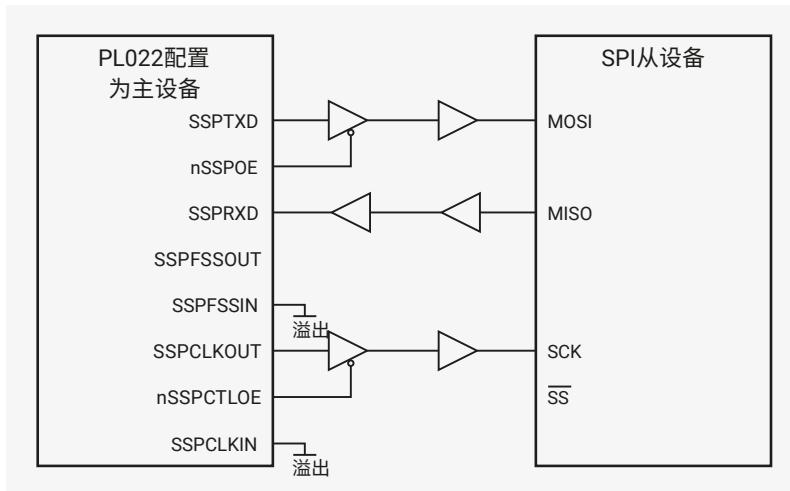
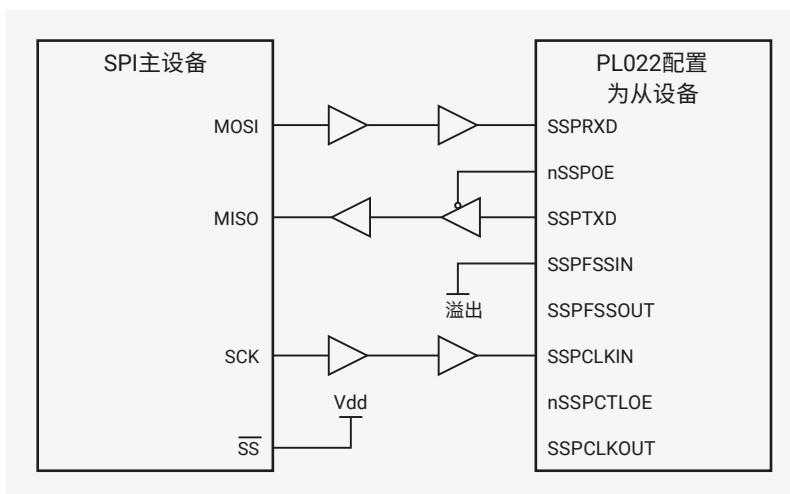


图105显示一个配置为主设备的摩托罗拉SPI，与配置为从设备的PrimeCell SSP（PL022）实例相连接。在此情况下，从设备选择信号（SS）永久拉高，以配置为主设备。主设备可通过SPI MOSI线向从设备广播，作为响应，从设备将其nSSPOE信号拉低，从而使其 SSPTXD数据输出到主设备的 MISO线。

图105。SPI主设备  
连接至PrimeCell  
SSP从设备



#### 12.3.4.16. PrimeCell DMA接口

PrimeCell SSP提供了连接DMA控制器的接口。PrimeCell SSP DMA控制寄存器SSPDMA<sub>R</sub>控制PrimeCell SSP的DMA操作。

DMA接口包括以下接收信号：

##### SSPRXDMASREQ

单字符DMA传输请求，由SSP发出。当接收FIFO中至少有一个字符时，该信号被激活。

##### SSPRXDMABREQ

突发DMA传输请求，由SSP发出。当接收FIFO中包含四个或更多字符时，该信号被激活。

##### SSPRX DMA清除

DMA请求清除信号，由DMA控制器发出，用以清除接收请求信号。若请求DMA突发传输，则在突发传输最后一条数据传送时，清除信号被发送。

DMA接口包括以下发送信号：

**SSPTXDMASREQ**

单字符DMA传输请求，由SSP发出。当发送FIFO中至少有一个空位时，该信号被激活。

**SSPTXDMABREQ**

突发DMA传输请求，由SSP发出。当发送FIFO中包含四个或更少字符时，该信号被激活。

**SSPTX DMA清除**

DMA请求清除信号由DMA控制器断言，用以清除传输请求信号。若请求DMA突发传输，则在突发传输的最后一数据传输期间断言清除信号。

突发传输请求信号与单次传输请求信号并非相互排斥。两者可同时断言。例如，当接收FIFO中的数据量超过四的水印水平时，突发传输请求信号与单次传输请求信号将同时被断言。当接收FIFO中剩余数据量低于水印水平时，仅断言单次请求信号。此情形适用于流中剩余待接收字符数少于一次突发的情况。

例如，如需接收19个字符，DMA控制器会先进行四次四字符的突发传输，随后进行三次单次传输以完成数据流。

**注意**

对于剩余的三个字符，PrimeCell SSP不会断言突发传输请求信号。

各请求信号在相应DMA清除信号断言前，均保持断言状态。在请求清除信号取消断言后，请求信号可根据前述章节描述的条件重新激活。若PrimeCell SSP被禁用或DMA使能信号被清除，所有请求信号将被取消断言。

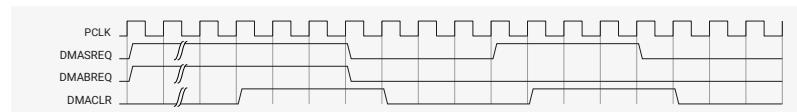
表1098显示了传输和接收FIFO的 DMABREQ 触发点。

表1098。发送和接收FIFO的DMA触发点

突发长度		
水位线水平	传输端，空闲位置数量	接收端，已填充位置数量
1/2	4	4

图106展示了单次传输请求与突发传输请求的时序图，以及相应的DMA清除信号。所有信号均同步于PCLK。

图106。DMA传输波形



### 12.3.5. 寄存器列表

SPI0和SPI1寄存器的基地址分别为0x40080000和0x40088000（在SDK中定义为SPI0\_BASE和SPI1\_BASE）。

表1099。SPI寄存器列表

偏移量	名称	说明
0x000	SSPCR0	控制寄存器 0，SSPCR0，见第3-4页
0x004	SSPCR1	控制寄存器 1，SSPCR1，见第3-5页
0x008	SSPDR	数据寄存器，SSPDR，见第3-6页
0x00c	SSPSR	状态寄存器，SSPSR，见第3-7页
0x010	SSPCPSR	时钟预分频寄存器，SSPCPSR，见第3-8页

偏移量	名称	说明
0x014	<a href="#">SSPIMSC</a>	中断屏蔽设置或清除寄存器, SSPIMSC, 见第3-9页
0x018	<a href="#">SSPRIS</a>	原始中断状态寄存器, SSPRIS, 见第3-10页
0x01c	<a href="#">SSPMIS</a>	屏蔽中断状态寄存器, SSPMIS, 见第3-11页
0x020	<a href="#">SSPICR</a>	中断清除寄存器, SSPICR, 见第3-11页
0x024	<a href="#">SSPDMACR</a>	DMA控制寄存器, SSPDMACR, 见第3-12页
0xfe0	<a href="#">SSPPERIPHIDO</a>	外设识别寄存器, SSPPERIPHIDO-3, 见第3-13页
0xfe4	<a href="#">SSPPERIPHID1</a>	外设识别寄存器, SSPPERIPHID0-3, 见第3-13页
0xfe8	<a href="#">SSPPERIPHID2</a>	外设识别寄存器, SSPPERIPHID0-3, 见第3-13页
0xfec	<a href="#">SSPPERIPHID3</a>	外设识别寄存器, SSPPERIPHID0-3, 见第3-13页
0xff0	<a href="#">SSPPCELLID0</a>	PrimeCell识别寄存器, SSPPCELLID0-3, 见第3-16页
0xff4	<a href="#">SSPPCELLID1</a>	PrimeCell识别寄存器, SSPPCELLID0-3, 见第3-16页
0xff8	<a href="#">SSPPCELLID2</a>	PrimeCell识别寄存器, SSPPCELLID0-3, 见第3-16页
0ffc	<a href="#">SSPPCELLID3</a>	PrimeCell识别寄存器, SSPPCELLID0-3, 见第3-16页

## SPI: SSPCR0 寄存器

偏移: 0x000

### 描述

控制寄存器 0, SSPCR0, 见第3-4页

表1100. SSPCR0  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:8	<b>SCR</b> : 串行时钟速率。SCR的值用于生成PrimeCell SSP的发送和接收比特率。比特率计算公式为: $F_{SSPCLK} = CPSDVSR \times (1+SCR)$ , 其中CPSDVSR为通过SSPCPSR寄存器设定的2至254之间的偶数, SCR取值范围为0至255。	读写	0x00
7	<b>SPH</b> : SSPCLKOUT 相位, 仅适用于 Motorola SPI 帧格式。详见第 2-10 页 Motorola SPI 帧格式。	读写	0x0
6	<b>SPO</b> : SSPCLKOUT 极性, 仅适用于 Motorola SPI 帧格式。详见第 2-10 页 Motorola SPI 帧格式。	读写	0x0
5:4	<b>FRF</b> : 帧格式: 00 Motorola SPI 帧格式; 01 TI 同步串行帧格式; 10 National Semiconductor Microwire 帧格式; 11 保留, 未定义操作。	读写	0x0
3:0	<b>DSS</b> : 数据大小选择: 0000 保留, 未定义操作; 0001 保留, 未定义操作; 0010 保留, 未定义操作; 0011 4 位数据。 0100 5 位数据; 0101 6 位数据; 0110 7 位数据; 0111 8 位数据; 1000 9 位数据; 1001 10 位数据; 1010 11 位数据; 1011 12 位数据; 1100 13 位数据; 1101 14 位数据; 1110 15 位数据; 1111 16 位数据。	读写	0x0

## SPI: SSPCR1 寄存器

偏移: 0x004

### 说明

控制寄存器 1, SSPCR1, 见第3-5页

表 1101. SSPCR1  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>SOD</b> : 从模式输出禁用。此位仅在从模式 (MS=1) 下有效。在多从设备系统中, PrimeCell SSP 主设备可向系统内所有从设备广播消息, 同时确保仅有 一台从设备驱动其串行输出线上的数据。在此类系统中, 多个从设备的 RXD 线可共用连线。为使系统正常运行, 当 PrimeCell SSP 从设备不应驱动 SSPTXD 线上信号时, 可置位 SOD 位: 0 表示 SSP 可在从模式下驱动 SSPTXD 输出; 1 表示 SSP 在从模式下不得驱动 SSPTXD 输出。	读写	0x0
2	<b>MS</b> : 主/从模式选择。此位仅在 PrimeCell SSP 被禁用时 (SSE=0) 允许修改: 0 表示设备配置为主设备 (默认); 1 表示设备配置为从设备。	读写	0x0
1	<b>SSE</b> : 同步串行端口使能: 0 表示 SSP 操作禁用; 1 表示 SSP 操作启用。	读写	0x0
0	<b>LBM</b> : 环回模式: 0 启用正常串行端口操作; 1 传输串行移位器的输出在内部连接至接收串行移位器的输入。	读写	0x0

## SPI: SSPDR 寄存器

偏移量: 0x008

### 说明

数据寄存器, SSPDR, 见第3-6页

表 1102. SSPDR  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<b>DATA</b> : 发送/接收FIFO: 读取接收FIFO, 写入发送FIFO。当PrimeCell SSP 设置的数据位宽小于16位时, 必须将数据右对齐。发送逻辑忽略最高位的未使用位。 接收逻辑自动进行右对齐。	RWF	-

## SPI: SSPSR 寄存器

偏移: 0x00c

### 描述

状态寄存器, SSPSR, 见第3-7页

表 1103. SSPSR  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>BSY</b> : PrimeCell SSP忙标志, RO: 0 表示SSP空闲; 1 表示SSP当前正在传输和/或接收帧, 或发送FIFO非空。	只读	0x0
3	<b>RFF</b> : 接收FIFO满标志, RO: 0 表示接收FIFO未满; 1 表示接收FIFO已满。	只读	0x0
2	<b>RNE</b> : 接收FIFO非空, RO: 0 接收FIFO为空。1 接收FIFO非空。	只读	0x0
1	<b>TNF</b> : 发送FIFO未满, RO: 0 发送FIFO已满。1 发送FIFO未满。	只读	0x1
0	<b>TFE</b> : 发送FIFO为空, RO: 0 发送FIFO非空。1 发送FIFO为空。	只读	0x1

## SPI: SSPCPSR寄存器

偏移量: 0x010

### 描述

时钟预分频寄存器，SSPCPSR，见第3-8页

表1104. SSPCPSR  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>CPSDVSR</b> : 时钟预分频除数。必须是2至254之间的偶数，具体取决于SPCLK的频率。读取时最低有效位始终返回零。	读写	0x00

## SPI: SSPIMSC寄存器

偏移量: 0x014

### 描述

中断屏蔽设置或清除寄存器，SSPIMSC，见第3-9页

表1105. SSPIMSC  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>TXIM</b> : 发送FIFO中断屏蔽，0表示发送FIFO半空或更空条件中断被屏蔽，1表示发送FIFO半空或更空条件中断未被屏蔽。	读写	0x0
2	<b>RXIM</b> : 接收FIFO中断屏蔽，0表示接收FIFO半满或更少条件中断被屏蔽，1表示接收FIFO半满或更少条件中断未被屏蔽。	读写	0x0
1	<b>RTIM</b> : 接收超时中断掩码：0表示接收FIFO非空且超时期间无读取的中断被掩盖。1表示接收FIFO非空且超时期间无读取的中断未被掩盖。	读写	0x0
0	<b>RORIM</b> : 接收溢出中断掩码：0表示接收FIFO满时写入的中断被掩盖。1表示接收FIFO满时写入的中断未被掩盖。	读写	0x0

## SPI: SSPRIS 寄存器

偏移量: 0x018

### 描述

原始中断状态寄存器，SSPRIS，见第3-10页

表1106. SSPRIS  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>TXRIS</b> : 表示SSPTXINTR中断掩码前的原始中断状态	只读	0x1
2	<b>RXRIS</b> : 表示SSPRXINTR中断掩码前的原始中断状态	只读	0x0
1	<b>RTRIS</b> : 表示SSPRTINTR中断掩码前的原始中断状态	只读	0x0

位	描述	类型	复位
0	<b>RORRIS:</b> 表示SSPRORINTR中断掩码前的原始中断状态	只读	0x0

## SPI: SSPMIS 寄存器

偏移量: 0x01C

### 描述

屏蔽中断状态寄存器, SSPMIS, 见第3-11页

表1107. SSPMIS  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>TXMIS:</b> 给出掩盖后的传输FIFO掩码中断状态, 属于SSPTXINTR中断	只读	0x0
2	<b>RXMIS:</b> 给出掩盖后的接收FIFO掩码中断状态, 属于SSPRXINTR中断	只读	0x0
1	<b>RTMIS:</b> 给出掩盖后的接收超时掩码中断状态, 属于SSPRTINTR中断	只读	0x0
0	<b>RORMIS:</b> 给出掩盖后的接收溢出掩码中断状态, 属于SSPRORINTR中断	只读	0x0

## SPI: SSPICR寄存器

偏移量: 0x020

### 描述

中断清除寄存器, SSPICR, 见第3-11页

表1108. SSPICR  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RTIC:</b> 清除SSPRTINTR中断	WC	0x0
0	<b>RORIC:</b> 清除SSPRORINTR中断	WC	0x0

## SPI: SSPDMACR寄存器

偏移量: 0x024

### 说明

DMA控制寄存器, SSPDMACR, 见第3-12页

表1109.  
SSPDMACR寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>TXDMAE:</b> 传输DMA使能。若该位设为1, 则启用传输FIFO的DMA。	读写	0x0
0	<b>RXDMAE:</b> 接收DMA使能。若该位设为1, 则启用接收FIFO的DMA。	读写	0x0

## SPI: SSPPERIPHIDO 寄存器

偏移: 0xfe0

**描述**

外设识别寄存器，SSPPeriphID0-3，见第3-13页

表 1110。  
SSPPERIPHID0  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>PARTNUMBER0</b> : 这些位的回读值为 0x22	只读	0x22

**SPI: SSPPERIPHID1 寄存器**

偏移量: 0xfe4

**描述**

外设识别寄存器，SSPPeriphID0-3，见第3-13页

表 1111。  
SSPPERIPHID1  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:4	<b>DESIGNERO</b> : 这些位读取值为 0x1	只读	0x1
3:0	<b>PARTNUMBER1</b> : 这些位读取值为 0x0	只读	0x0

**SPI: SSPPERIPHID2 寄存器**

偏移量: 0xfe8

**描述**

外设识别寄存器，SSPPeriphID0-3，见第3-13页

表 1112。  
SSPPERIPHID2  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:4	<b>REVISION</b> : 这些位返回外围设备的修订版本	只读	0x3
3:0	<b>DESIGNER1</b> : 这些位读取值为 0x4	只读	0x4

**SPI: SSPPERIPHID3 寄存器**

偏移量: 0fec

**描述**

外设识别寄存器，SSPPeriphID0-3，见第3-13页

表 1113。  
SSPPERIPHID3  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>配置</b> : 这些位读取值为 0x00	只读	0x00

**SPI: SSPPCELLID0 寄存器**

偏移量: 0xff0

**描述**

PrimeCell识别寄存器，SSPCellID0-3，见第3-16页

表1114。  
SSPPCELLID0 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>SSPPCELLID0</b> : 这些位读取结果为0x0D	只读	0x0d

## SPI: SSPPCELLID1 寄存器

偏移: 0xff4

### 描述

PrimeCell识别寄存器, SSPCellID0-3, 见第3-16页

表1115。  
SSPPCELLID1 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>SSPPCELLID1</b> : 这些位读取结果为0xF0	只读	0xf0

## SPI: SSPPCELLID2 寄存器

偏移: 0xff8

### 描述

PrimeCell识别寄存器, SSPCellID0-3, 见第3-16页

表1116。  
SSPPCELLID2 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>SSPPCELLID2</b> : 这些位读取结果为0x05	只读	0x05

## SPI: SSPPCELLID3 寄存器

偏移: 0ffc

### 描述

PrimeCell识别寄存器, SSPCellID0-3, 见第3-16页

表1117。  
SSPPCELLID3 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	<b>SSPPCELLID3</b> : 这些位读取值为0xB1	只读	0xb1

## 12.4. ADC 与温度传感器

RP2350内置的模数转换器（ADC）具备以下特性：

- SAR ADC（参见第12.4.3节）
- 500 kS/s（使用独立的48 MHz时钟）
- 12位分辨率, 9.2 ENOB（参见第12.4.4节）
- 五路或九路输入多路复用器：
  - QFN-60封装引脚上有四个输入, 与GPIO[29:26]共用
  - QFN-80封装引脚上有八个输入, 与GPIO[47:40]共用
  - 一个输入专用于内部温度传感器（参见第12.4.6节）

- 八级接收采样FIFO
- 中断生成
- DMA接口（参见第12.4.3.5节）

图107展示了QFN-60封装中ADC通道的排列。图108展示了QFN-80封装中相同的排列。

图107。QFN-60的ADC连接图。  
该封装具有四个外部ADC输入（  
编号0至3），对应Bank 0中GPIO 26至29  
。内部温度  
传感器连接至第五通道（通道4  
）。其功能与RP2040的ADC配置  
相同，尽管底层硬件不同，以支持  
QFN-80上的额外通道。  
通道。

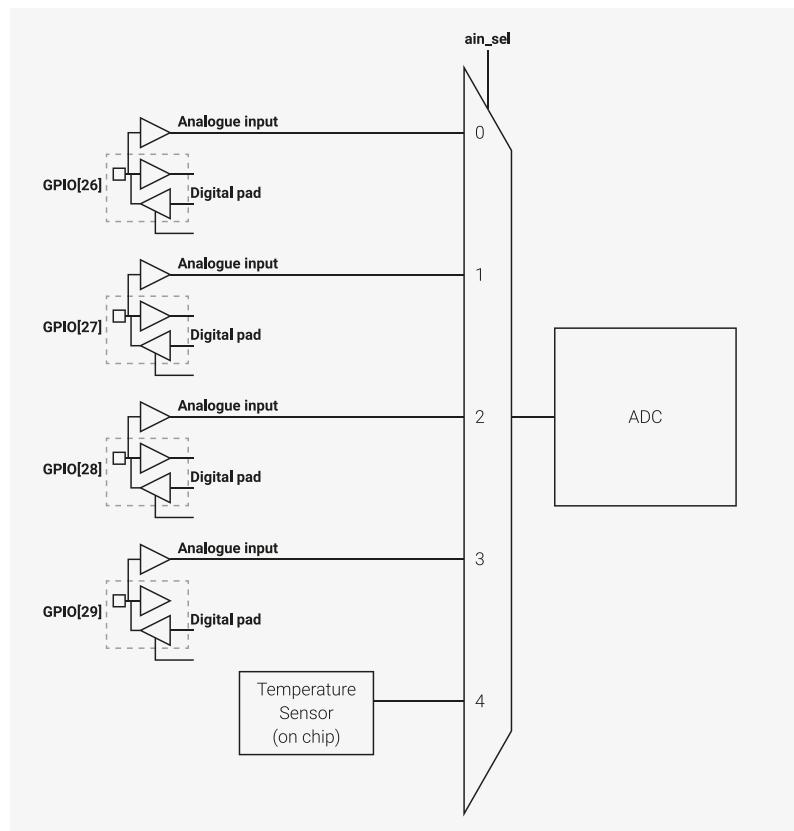
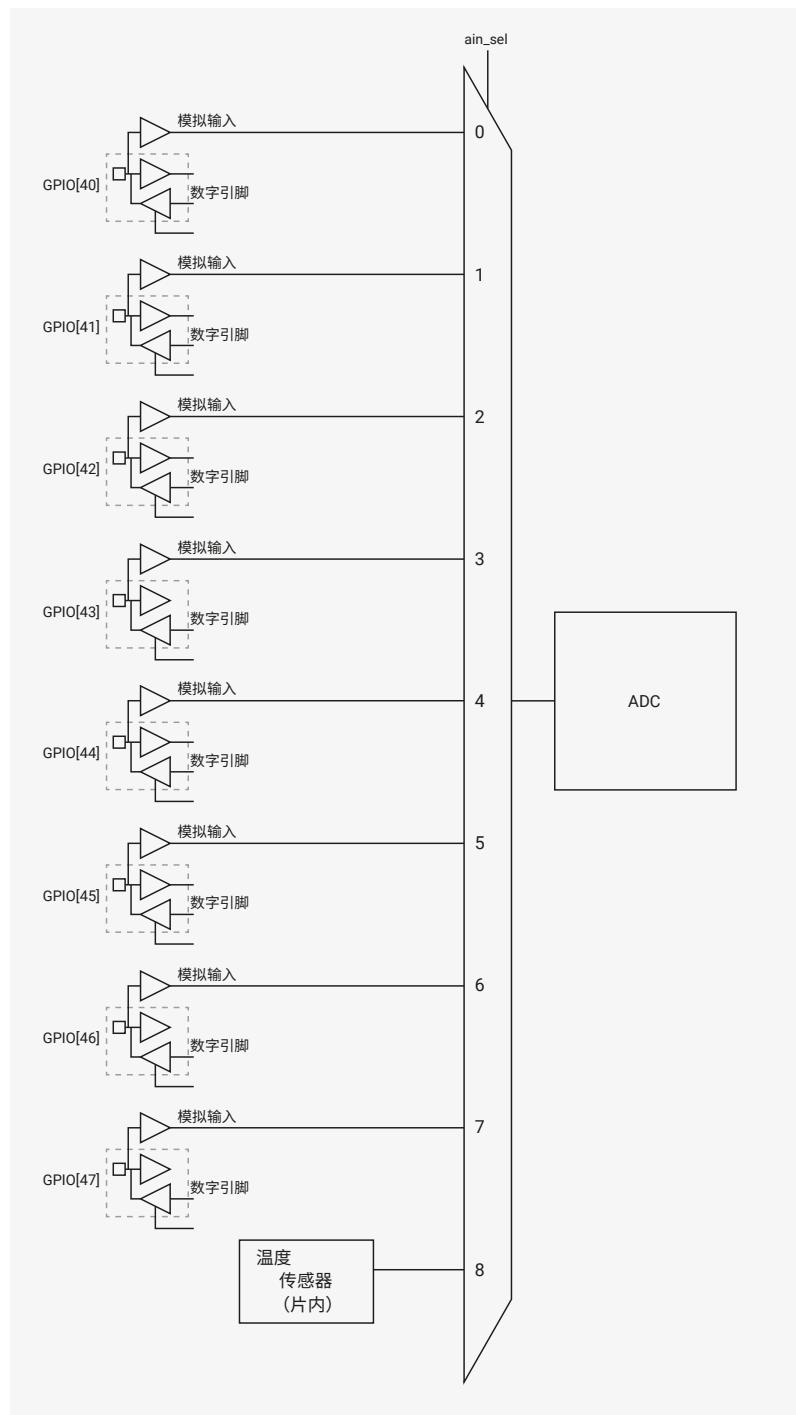


图108。QFN-80 ADC连接图。该封装提供八个外部ADC输入（通道0至7），位于Bank 0的GPIO 40至47引脚。内部温度传感器连接至第九通道（通道8）。与QFN-60类似，每个ADC输入与Bank 0的数字GPIO共用封装引脚：

通常，当ADC正在使用时，数字功能会被禁用。



使用与GPIO引脚共用的ADC输入时，务必通过在引脚的管脚控制寄存器中将 IE置低且将 OD置高，以禁用该引脚的数字功能。详情请参见第9.11.3节，“管脚控制 - 用户组”。

最大ADC输入电压由数字IO电源电压（**IOVDD**）决定，非ADC电源电压。

（**ADC\_AVDD**）。例如，若 **IOVDD**电源电压为1.8 V，则ADC输入端电压不应超过1.8 V + 10%，即使ADC\_AVDD供电为3.3 V。电压超出 **IOVDD**范围将导致通过ESD保护二极管的漏电流。详情请参见第14.9节，“电气规格”。

### 12.4.1. 与 RP2040 的变化

- 如勘误表RP2040-E11所示，已消除码0x200、0x600、0xa00及0xe00处差分非线性中的尖峰，ADC精度提升约0.5 ENOB。

- 仅在QFN-80封装中，外部ADC输入通道数量由4个增加至8个。

### 12.4.2. ADC 控制器

数字控制器负责管理RP2350 ADC的操作细节，并提供以下附加功能：

- 单次触发或自由运行采样模式
- 具有DMA接口的采样FIFO
- 定时器（16位整数部分，8位小数部分）用于设置自由运行采样速率
- 自由运行采样模式下的多通道循环采样
- 自由运行采样模式中，采样数据可选择右移至8位，以便通过DMA传输至系统字节缓冲区内存

#### 12.4.2.1 通道连接

QFN-60中ADC通道对应的GPIO连接如下

表1118 QFN-60  
封装上的ADC通道  
连接

通道	连接
0	GPIO[26]
1	GPIO[27]
2	GPIO[28]
3	GPIO[29]
4	温度传感器

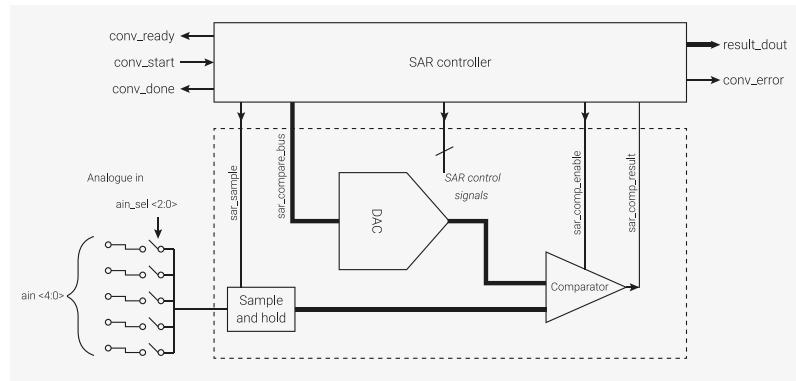
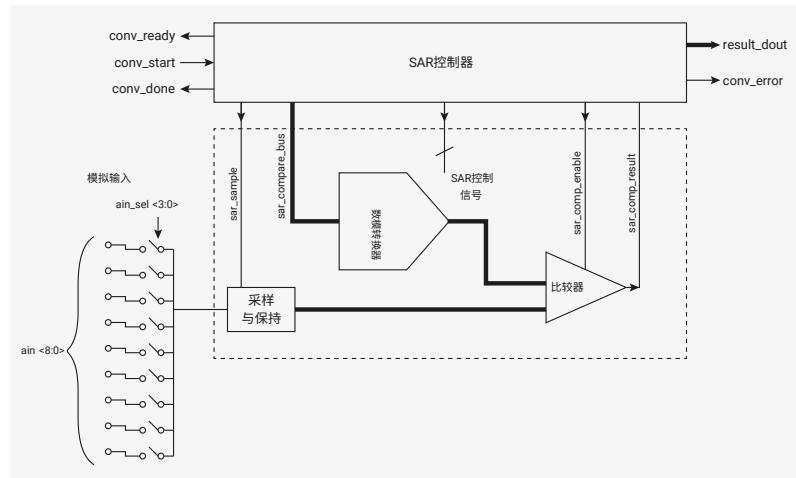
ADC通道连接至QFN-80封装中的以下GPIO端口

表1119。QFN-  
80封装上的ADC通  
道连接

通道	连接
0	GPIO[40]
1	GPIO[41]
2	GPIO[42]
3	GPIO[43]
4	GPIO[44]
5	GPIO[45]
6	GPIO[46]
7	GPIO[47]
8	温度传感器

### 12.4.3. SAR ADC

逐次逼近寄存器模拟-数字转换器（SAR ADC）是数字控制器与模拟电路的组合，如图109和图110所示。

图109。SAR ADC  
方框图 QFN-60图110。SAR ADC  
方框图 QFN-80

ADC需要一个48 MHz时钟（`clk_adc`），该时钟可由USB PLL提供。采样过程需96个时钟周期（ $96 \times 1/48 \text{ MHz} = 2\text{微秒}$ 每样本（500 kS/s）。启用ADC前，必须正确设置时钟信号。

当ADC模块提供时钟且复位已解除时，向CS.EN写入1将启动ADC模拟硬件的短暂内部上电序列。经过几个时钟周期后，CS.READY将置高，表示ADC已准备好开始首次转换。

为节省功耗，您可随时通过清除CS.EN来禁用ADC。CS.EN不启用温度传感器偏置源；该偏置源由独立控制，详情请参见第12.4.6节。

ADC输入端为电容性。采样时，ADC在输入端并联约1pF电容。封装、PCB布线及其他外部因素会引入额外电容。即使在500 kS/s采样速率下，有效阻抗仍超过100 kΩ。直流测量无需缓冲。

### 12.4.3.1. 单次采样

要选择ADC输入，请写入CS.AINSEL：

- 在QFN-60封装上，有4个外部输入，AINSEL值为0→3对应GPIO26→GPIO29上的ADC输入。将AINSEL设置为4即选用内部温度传感器。
- 在QFN-80封装上，有8个外部输入，AINSEL值为0→7对应GPIO40→GPIO47上的ADC输入。将AINSEL设置为8即选用内部温度传感器。

切换AINSEL无需等待稳定时间。

向CS.START\_ONCE写入1以立即启动一次新的转换。CS.READY会拉低，表示转换正在进行。经过96个`clk_adc`周期后，CS.READY会拉高。12位转换结果可在RESULT中读取。

### 12.4.3.2. 自由运行采样模式

设置 CS.START\_MANY 后，ADC 会自动按照固定间隔启动新转换。最新的转换结果始终保存在RESULT中，但对于IRQ或DMA驱动的样本流，必须启用ADC FIFO（参见第12.4.3.4节）。

默认情况下（DIV = 0），新转换在前一次转换结束后立即启动，每96个周期产生一个新样本。在48 MHz时钟频率下，此设置产生500 kS/s的采样率。

将DIV.INT设置为正值 n以每 n+ 1个周期触发一次ADC转换。如果当前转换正在进行，ADC将忽略该设置，因此通常 n应≥ 96。例如，以48 MHz时钟运行时，将DIV.INT设置为47999可使ADC以1 kS/s速度运行。

节拍定时器支持分数速率分频（一阶增量调制）。当将DIV.FRAC设置为非零值时，

$$\text{ADC平均每 } \frac{1 + \text{INT} + \frac{\text{FRAC}}{256}}{256} \text{ 个周期启动一次新转换，通过改变采样间隔实现。}$$

介于 INT + 1 and INT + 2.

### 12.4.3.3. 采样多个输入

CS.RROBIN 允许 ADC 在执行自由运行采样时以轮询方式采样多个输入。

RROBIN 的每个位对应 CS.AINSEL 的五个可能值之一。当 ADC 完成转换后，CS.AINSEL 会自动切换至 RROBIN 中对应位被设置的下一个输入通道。

若要禁用轮询采样功能，请向 CS.RROBIN 写入全零值。

例如，若AINSEL 初始值为 0，且 RROBIN 设置为 0x06（即第1和第2位被设置），则 ADC 会按以下顺序采样通道：

1. 通道 0
2. 通道 1
3. 通道 2
4. 通道 1
5. 通道 2
6. 通道 1
7. 通道 2

ADC 会无限期持续采样通道 1 和通道 2。

#### 注意

AINSEL的初始值无需与RROBIN中的置位位对应。

### 12.4.3.4. FIFO示例

您可以直接从RESULT寄存器读取ADC样本，或将其存储在本地8条目的FIFO中，并从FIFO读取。使用FCS寄存器控制FIFO操作。

当FCS.EN被设置时，ADC会将每次转换结果写入FIFO。ADC通过 IRQ或 DREQ信号通知时，软件中断处理程序或RP2350 DMA可以从FIFO读取该样本。或者，软件可通过轮询FCS中的状态位等待每个样本可用。

若转换完成时FIFO已满，粘性错误标志FCS.OVER将被置位。FIFO满载时，当前内容保持不变，因此在此期间完成的任何转换结果将丢失。

有两个标志用于控制ADC写入FIFO的数据：

- FCS.SHIFT将FIFO数据右移至8位大小（即FIFO的位7:0对应转换结果的位11:4）。该功能适用于向内存中的字节缓冲区执行8位DMA传输，从而实现更深层次的捕获缓冲区，代价是牺牲部分精度。

- FCS.ERR会设置每个FIFO值的FIFO.ERR标志，表明发生了转换错误，即SAR未能收敛。

转换错误表示一个或多个位的比较未能在规定时间内完成。转换错误通常由比较器亚稳态引起：输入信号越接近比较器阈值，做出决策所需时间越长。比较器增益越高，转换错误的概率越低。

#### 警告

由于转换错误会导致未定义结果，您应始终丢弃包含转换错误的样本。

#### 12.4.3.5. DMA

RP2350 DMA（见第12.6节）能够通过对FIFO寄存器执行正常的内存映射读取，从样本FIFO中获取ADC样本，该读取由 **AD\_C\_DREQ** 系统数据请求信号控制。在使用DMA获取ADC样本之前，您必须：

- 启用样本FIFO（FCS.EN），以便将样本写入其中；FIFO默认处于禁用状态，以防止在ADC用于单次转换时意外填满。在启动ADC之前，配置ADC采样率（第12.4.3.2节）。
- 通过FCS.DREQ\_EN启用ADC的数据请求握手（**DREQ**）。
- 在所用的DMA通道中，选择 **DREQ\_ADC** 数据请求信号（第12.6.4.1节）。
- 将 **DREQ** 断言阈值（FCS.THRESH）设置为1，使DMA在FIFO中存在单个样本时立即传输。此阈值同样用于IRQ断言，因此非DMA应用场景可能偏好设置较高值以降低中断频率。
- 如果DMA传输大小设置为8位（即DMA传输至内存中的字节数组），则需设置FCS.SHIFT以预移位FIFO样本至8位有效位。
- 要对多个输入通道进行采样，请将这些通道的掩码写入 CS.RROBIN。同时，使用 CS.AINSEL 选择第一个采样通道。

ADC配置完成后，先启动DMA通道，然后通过 CS.START\_MANY 启动ADC转换。DMA传输完成后，如采样结束，可停止ADC；否则，应在 FIFO 填满前立即启动新一轮DMA传输。清除 CS.START\_MANY 以停止ADC后，软件须轮询 CS.REady 以确认最后一次转换已完成，并清空 FIFO 中的任何残留样本。

#### 12.4.3.6. 中断

使用 INTE 以便在 FIFO 水平达到 FCS.THRESH 所定义的阈值时生成中断。

使用 INTS 读取中断状态。要清除中断，必须将 FIFO 排空至低于 FCS.THRESH 的水平。

#### 12.4.3.7. 电源

RP2350 将 ADC 电源引脚分离出来，以便进行噪声滤波。

#### 12.4.4. ADC ENOB

ADC 的有效位数（ENOB）详细信息见表 1438。

#### 12.4.5. INL 与 DNL

详细信息将随后提供。

#### 12.4.6. 温度传感器

温度传感器测量偏置双极性二极管的  $V_{be}$  电压，该二极管连接至 QFN-60 的第五 ADC 通道 ( $A_{INSEL}=4$ ) 或 QFN-80 的第九 ADC 通道 ( $A_{INSEL}=8$ )。通常， $V_{be}=0.706\text{ V}$  ( $27^\circ\text{C}$ )，斜率为每摄氏度  $-1.721\text{ mV}$ 。因此，温度（单位： $^\circ\text{C}$ ）可近似表示为：

$$T = 27 - \frac{(\text{ADC\_voltage} - 0.706)}{0.001721}$$

由于  $V_{be}$  及其斜率在不同温度范围内及不同器件间可能存在变化，如需准确测量，可能需要用户校准。

使用前必须通过 CS.TS\_EN 使能温度传感器的偏置电源，此举会使  $\text{ADC\_AVDD}$  的电流消耗增加约  $40\mu\text{A}$ 。

##### ① 注意

板载温度传感器对参考电压误差极为敏感。在  $3.3\text{ V}$  参考电压下，ADC 返回值 891 对应温度  $20.1^\circ\text{C}$ 。若参考电压比  $3.3\text{ V}$  低 1%，相同读数 891 对应温度将变为  $24.3^\circ\text{C}$ ，温差超过  $4^\circ\text{C}$ 。为提高内部温度传感器的准确度，建议添加外部参考电压。

#### 12.4.7. 寄存器列表

ADC 寄存器起始地址为  $0x400a0000$ （在 SDK 中定义为  $\text{ADC\_BASE}$ ）。

表1120.  
ADC寄存器列表

偏移量	名称	说明
0x00	CS	ADC控制与状态
0x04	RESULT	最近一次ADC转换结果
0x08	FCS	FIFO控制与状态
0x0c	FIFO	转换结果FIFO
0x10	DIV	时钟分频器。若非零，CS_START_MANY将在固定间隔而非连续启动转换。 写入任一字段时，分频器将被复位。 总周期为 $1 + \text{INT} + \text{FRAC} / 256$
0x14	中断	原始中断
0x18	INTE	中断使能
0x1c	INTF	中断触发
0x20	INTS	掩码与强制后的中断状态

## ADC: CS寄存器

偏移: 0x00

### 描述

ADC控制与状态

表1121.  
CS寄存器

位	描述	类型	复位
31:25	保留。	-	-
24:16	<b>RROBIN:</b> 循环采样。每通道1位。将所有位设为0以禁用。 否则，ADC 将以轮询方式循环读取每个启用的通道。  首次采样的通道为当前由AINSEL指示的通道。 每次转换后，AINSEL会更新为新选定的通道。	读写	0x000
15:12	<b>AINSEL:</b> 选择模拟多路复用输入。在轮询模式下自动更新。 此处针对封装选项进行了修正，确保仅提供已封装的ADC通道，且顺序正确。	读写	0x0
11	保留。	-	-
10	<b>ERR_STICKY:</b> 之前的某次ADC转换发生错误。写入1以清除。	WC	0x0
9	<b>ERR:</b> 最近一次ADC转换发生错误；结果未定义或存在噪声。	只读	0x0
8	<b>READY:</b> ADC准备开始新转换时为1。表示任意先前的转换已经完成。  转换进行中时为0。	只读	0x0
7:4	保留。	-	-
3	<b>START_MANY:</b> 当该位为1时，连续执行转换。前一次转换完成后，立即开始新转换。	读写	0x0
2	<b>START_ONCE:</b> 启动单次转换，自动清除。若start_many被置位，则忽略该位。	SC	0x0
1	<b>TS_EN:</b> 上电温度传感器。1 - 启用，0 - 禁用。	读写	0x0
0	<b>EN:</b> 上电ADC并使能其时钟。 1 - 启用，0 - 禁用。	读写	0x0

## ADC: RESULT寄存器

偏移: 0x04

表1122. RESULT  
寄存器

位	描述	类型	复位
31:12	保留。	-	-
11:0	最近一次ADC转换结果	只读	0x000

## ADC: FCS寄存器

偏移: 0x08

### 描述

FIFO控制与状态

表1123. FCS  
寄存器

位	描述	类型	复位
31:28	保留。	-	-
27:24	<b>THRESH</b> : 当电平 $\geq$ 阈值时, DREQ/IRQ触发	读写	0x0
23:20	保留。	-	-
19:16	<b>LEVEL</b> : FIFO中当前等待的转换结果数	只读	0x0
15:12	保留。	-	-
11	<b>OVER</b> : FIFO溢出时为1。写1清除。	WC	0x0
10	<b>UNDER</b> : FIFO欠流时为1。写1清除。	WC	0x0
9	<b>满</b>	只读	0x0
8	<b>空</b>	只读	0x0
7:4	保留。	-	-
3	<b>DREQ_EN</b> : 为1时, 当FIFO有数据即触发DMA请求	读写	0x0
2	<b>ERR</b> : 为1时, FIFO中结果同时出现转换错误位	读写	0x0
1	<b>SHIFT</b> : 为1时, FIFO结果右移至一个字节大小。使能字节缓冲区DMA。	读写	0x0
0	<b>EN</b> : 若为1, 则在每次转换后将结果写入FIFO。	读写	0x0

## ADC: FIFO寄存器

偏移: 0x0c

### 描述

转换结果FIFO

表1124. FIFO  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>ERR</b> : 若该样本发生转换错误则为1。样本若发生移位, 则保持在相同位置。	RF	-
14:12	保留。	-	-
11:0	<b>VAL</b>	RF	-

## ADC: DIV寄存器

偏移: 0x10

### 描述

时钟分频器。若非零, 则CS\_START\_MANY将以固定间隔启动转换, 而非连续启动。

写入任一字段时, 分频器将被复位。

总周期为 $1 + \text{INT} + \text{FRAC} / 256$

表1125. DIV  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:8	<b>INT</b> : 时钟除数的整数部分。	读写	0x0000
7:0	<b>FRAC</b> : 时钟除数的小数部分。一阶增量-Σ调制。	读写	0x00

## ADC: INTR寄存器

偏移: 0x14

### 说明

原始中断

表1126。INTR  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>FIFO:</b> 当采样FIFO达到设定级别时触发。 该级别可通过FCS_THRESH字段进行编程设定。	只读	0x0

## ADC: INTF寄存器

偏移: 0x18

### 说明

中断使能

表1127。INTF  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>FIFO:</b> 当采样FIFO达到设定级别时触发。 该级别可通过FCS_THRESH字段进行编程设定。	读写	0x0

## ADC: INTS寄存器

偏移量: 0x1c

### 描述

中断强制

表1128。INTS  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>FIFO:</b> 当采样FIFO达到设定级别时触发。 该级别可通过FCS_THRESH字段进行编程设定。	读写	0x0

## ADC: INTS寄存器

偏移: 0x20

### 说明

掩码与强制后的中断状态

表1129。INTS  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>FIFO:</b> 当采样FIFO达到设定级别时触发。 该级别可通过FCS_THRESH字段进行编程设定。	只读	0x0

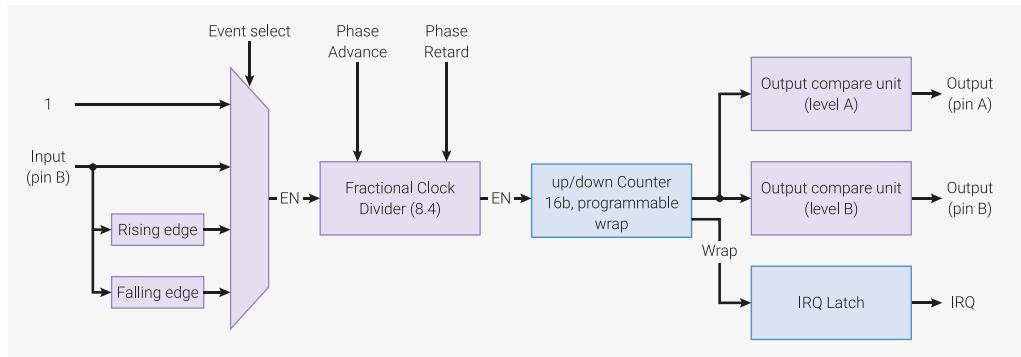
## 12.5. PWM

## 12.5.1. 概述

脉冲宽度调制（PWM）通过在规律间隔内使用受控宽度的正脉冲，平滑地调整数字信号的平均电压。高电平所占时间的比例称为占空比。这可用于近似模拟输出或控制开关式电源电子设备。

RP2350的PWM模块包含12个相同的切片。每个切片可驱动两个PWM输出信号，或测量输入信号的频率或占空比。每个切片的两个输出具有相同周期，但占空比独立变化，因此QFN-80封装中共计有24个可控PWM输出。

图111. 单个PWM切片。16位计数器从0计数至某个预设值后，根据PWM模式回绕至零，或反向计数。A和B输出根据当前计数值及预设的A和B阈值实现高低电平切换。计数器依据多种事件进行递增：可自由运行，亦可由B引脚输入信号的电平或边沿门控。分数除频器用于降低整体计数速率，以实现对输出频率的更精细控制。



每个 PWM 分片配备以下功能：

- 16 位计数器
- 8.4 分数时钟除频器
- 两个独立输出通道，占空比范围为 0% 至 100%（含）
- 双斜率及尾随边调制
- 用于频率测量的边沿敏感输入模式
- 用于占空比测量的电平敏感输入模式
- 可配置的计数器回绕值
  - 回绕与电平寄存器采用双缓冲设计，可在 PWM 运行时无竞争条件修改
- 计数器回绕时产生中断请求及 DMA 请求
- 运行期间可精确提前或延迟相位（按计数单位递增）

分片可通过单个全局控制寄存器同时启用或禁用分片随后以锁步模式运行，便于通过多个分片输出切换更复杂的电源电路。

### 12.5.1.1. 来自 RP2040 的变更

- 将切片数量从 8 个增加至 12 个，QFN-80 封装中新增的 4 个切片位于 GPIO 32 至 47。
- 新增第二条共享中断线（由 IRQ1\_INTE 控制），以辅助 PWM 切片作为简单重复计时器的使用。

## 12.5.2. 程序员模型

RP2350 上的所有 GPIO 引脚均可用于 PWM：

表 1130。RP2350 上 PWM 通道与 GPIO 引脚的映射。  
该映射亦显示于主 GPIO 功能表，表 646。

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM 通道	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PWM 通道	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
PWM 通道	8A	8B	9A	9B	10A	10B	11A	11B	8A	8B	9A	9B	10A	10B	11A	11B

- 前16个PWM通道（8个2通道切片）依次显示于GPIO0至GPIO15，顺序为 **PWM0 A**、**PWM0 B**、**PWM1 A**，依此类推。
- 该排列在GPIO16至GPIO31之间重复。GPIO16对应 **PWM0 A**，GPIO17对应 **PWM0 B**，依此类推至GPIO31的 **PWM7 B**。GPIO30及以上仅在QFN-80封装中支持。
- 剩余8个PWM通道（4个2通道切片）分布于GPIO32至GPIO39，并在GPIO40至GPIO47上重复出现。
- 如果您在两个GPIO引脚上选择相同的PWM输出，则两个引脚上的信号相同。
- 如果您使用 **B** 引脚作为输入并在多个GPIO引脚上选择该引脚，PWM切片将感知这两个GPIO输入的逻辑或。

### ① 注意

GPIO0至GPIO29的通道分配与RP2040相同，以保证引脚兼容性。这减少了RP2350 QFN-60封装选项中独立PWM输出的最大数量，但在此封装中，您仍可使用切片8至11实现定时器中断的重复。

#### 12.5.2.1. 脉宽调制 (PWM)

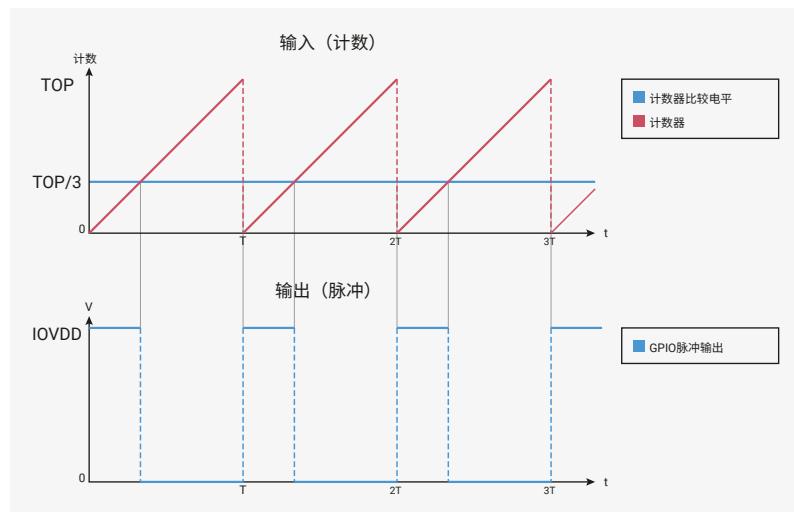
PWM硬件持续比较输入值与自由运行计数器的计数值。由此产生切换输出；高电平输出的持续时间与输入值相对应。高电平信号持续时间占总时间的比例称为信号的占空比。

计数周期由 **TOP**寄存器控制，最大周期为65536个计数周期，因计数器和 **TOP**寄存器均为16位。使用 **CC**寄存器配置输入值。

图112。计数器重复从0计数到TOP，形成锯齿波形。计数器持续与某输入值进行比较。当输入值高于计数器时，输出为高电平。

否则，输出为低电平。输出周期 $T$ 由计数器的TOP值及计数速度决定。

平均输出电压（占IO电源电压的比例）等于输入值除以计数周期 $(TOP + 1)$ 。



本示例演示了在RP2350的一个PWM切片上配置计数周期以及A、B计数比较电平。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pwm/hello\\_pwm/hello\\_pwm.c](https://github.com/raspberrypi/pico-examples/blob/master/pwm/hello_pwm/hello_pwm.c) 第14行至第29行

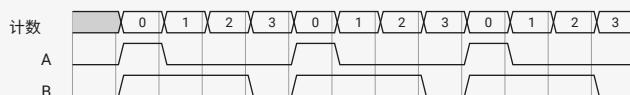
```

14 // 指示GPIO 0和1分配给PWM功能
15 gpio_set_function(0, GPIO_FUNC_PWM);
16 gpio_set_function(1, GPIO_FUNC_PWM);
17
18 // 查询连接到GPIO 0的PWM切片编号 (为切片0)
19 uint slice_num = pwm_gpio_to_slice_num(0);
20
21 // 设定计数周期为4 (含0至3)
22 pwm_set_wrap(slice_num, 3);
23 // 设置通道A输出高电平持续1个周期后降低
24 pwm_set_chan_level(slice_num, PWM_CHAN_A, 1);
25 // 在下降之前将初始B输出设置为高电平，持续三个周期
26 pwm_set_chan_level(slice_num, PWM_CHAN_B, 3);
27 // 启动PWM
28 pwm_set_enabled(slice_num, true);

```

图113展示了配置完成后的PWM硬件运行情况。

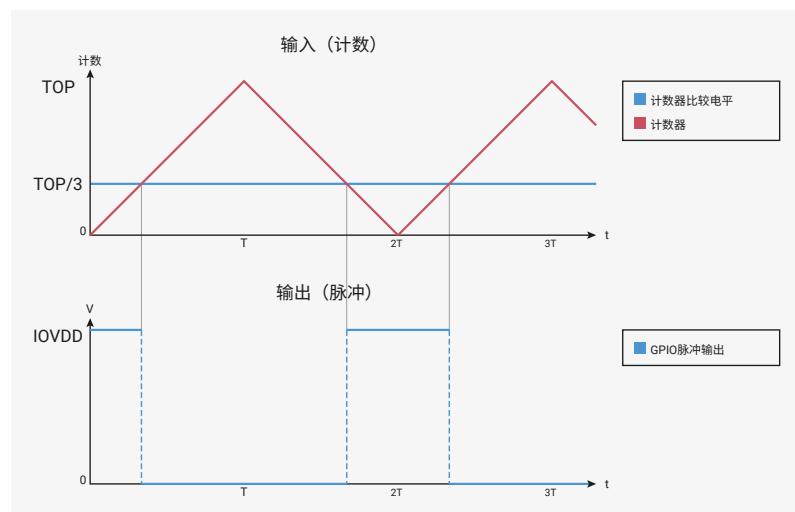
图113。该片段计数器反复从0计数至3，该值被设置为TOP。因此输出波形的周期为4。输出A在4个周期中高电平持续1个周期，故平均输出电压为IO供电电压的1/4。



默认情况下，PWM片段计数器向上计数，直到达到TOP寄存器的数值。达到TOP数值后，计数器回绕至0。或者，将CSR\_P\_H\_CORRECT设置为1以启用相位校正模式，在该模式下计数器在达到TOP后向下计数，直至再次达到0。

输出B在每4个周期中高电平持续3个周期。A和B的上升沿始终保持同步。

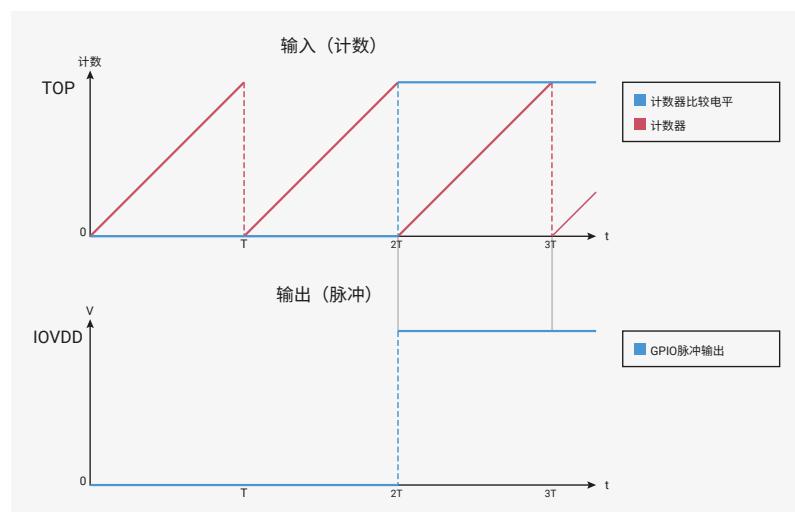
图114。在相位校正模式下，计数器达到TOP后，将从TOP向0倒计数。



### 12.5.2.2. 0%与100%占空比

RP2350 PWM能够产生无跳变的0%与100%占空比输出。

图115。当CC=0时，输出为无波动的0%占空比；当CC=TOP + 1时，输出为无波动的100%占空比。  
1



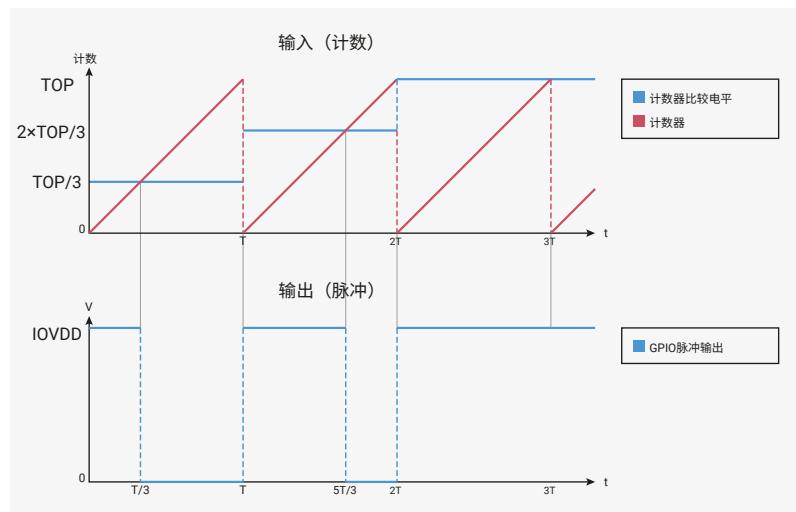
CC值为0产生0%输出：输出信号始终为低电平。CC值为TOP + 1（对应非相位校正时的周期）产生100%输出。如果 TOP为254，则计数器周期为255个计数，且 CC值在0至255（含）范围内将生成占空比在0%至100%（含）范围内。

在0%和100%时实现无毛刺输出，有助于避免开关损耗，例如当MOSFET被控制在其最小和最大电流等级时。

### 12.5.2.3. 双缓冲

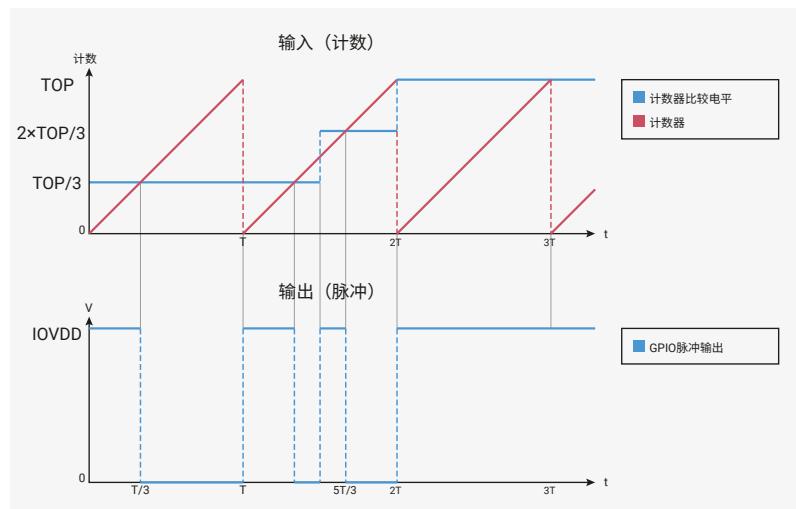
图116显示输入值变化如何引起输出占空比的变化。此方法可近似模拟波形，如正弦波。

图116。输入值随每个计数周期变化：依次为 $TOP/3$ 、 $2 \times TOP/3$ ，最后为 $TOP + 1$ ，对应100%的占空比。  
输入值的每次增加都会导致输出占空比的相应增加。



在图116中，输入值仅在计数器绕回至0的瞬间发生变化。图117展示了如果允许输入值在其他时间变化时，输出端将产生不期望的毛刺。

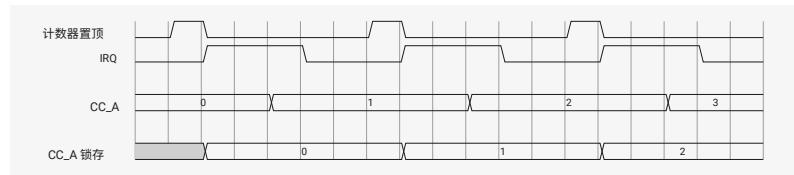
图117。输入值在计数器处于中间斜坡阶段时发生变化。这会导致输出端产生额外的跳变。



若同时修改 `TOP` 寄存器，行为将更加复杂难以分析。软件难以在准确时刻写入 `CC` 或 `TOP` 寄存器。为解决此问题，每个切片设有两份 `CC` 和 `TOP` 寄存器：一份供软件修改，另一份为内部副本，在计数器绕回时由第一份寄存器更新。软件可以自由修改自身寄存器副本，但 PWM 输出直到下一次绕回才应用这些更改。

图118展示了软件中断处理程序在每次计数器绕回时更改 `CC_A` 值的事件序列。

图118。每次计数器循环都会触发中断请求信号置位。处理器进入中断处理程序，写入其 CC 寄存器副本，并清除中断。



计数器再次循环时，**CC**寄存器的锁存副本会瞬间更新为软件写入的新数值，该数值控制下一周期的占空比。**IRQ**信号被重新置位，以便软件向其**CC**寄存器副本写入新的数值。

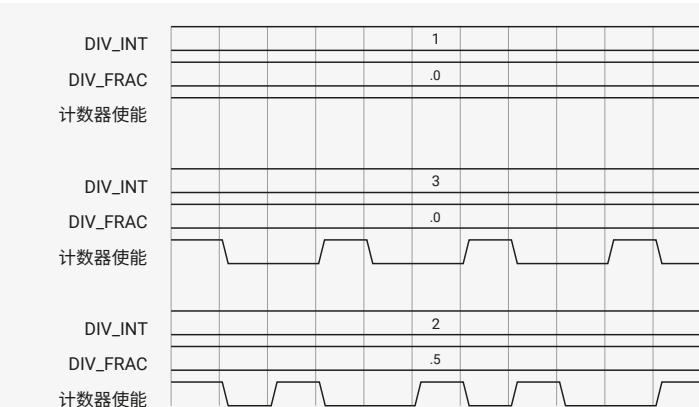
对 **CC** 或 **TOP** 的写入数值及时间不受限制。在正常 PWM 模式下（当 **CSR\_PH\_CORRECT** 为 0 时，锁存副本在计数器循环至 0 时更新，该过程每 **TOP+1** 个周期发生一次。在相位校正模式下（**CSR\_PH\_CORRECT** 为 1），锁存副本会在计数从 0 回到 0 的转换时更新，即计数器停止向下计数并开始向上计数时）。

#### 12.5.2.4. 时钟分频器

每个切片包含一个由 **DIV** 寄存器配置的 8 位整数部分和 4 位小数部分的分数时钟分频器。时钟分频器允许您将计数速率降低最多 256 倍。为此，PWM 会生成一个使能信号以控制计数器的运行。这使您能够实现远低于系统时钟的输出频率。例如，对于 125MHz 的系统时钟，时钟分频器可将计数速率降低至约 7.5Hz。低于此频率则需要系统定时器中断（参见第 12.8 节）。

图119。时钟分频器生成使能信号。计数器仅在该信号为高电平的周期内计数。分频值为 7 时，使能信号在每个周期均被断言，计数器在每个系统时钟周期内计数加一。更高的分频值会减少计数使能信号断言的频率。

分数除法通过拉长某些使能脉冲间隔，实现平均分数计数率。



分数分频器为一阶  $\Delta$ - $\Sigma$ 型。

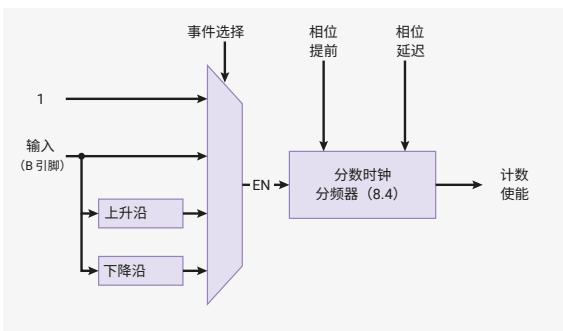
时钟分频器在采用电平敏感或边沿敏感模式进行占空比或频率测量时，可扩展有效计数范围。

#### 12.5.2.5. 电平敏感与边沿敏感触发

PWM 提供以下计数模式：

- 默认自由运行模式，片段启用时持续计数（自由运行）
- 当 B 引脚为高电平时持续计数（电平敏感）
- 对 B 引脚的每个上升沿计数一次（上升沿敏感）
- 对 B 引脚的每个下降沿计数一次（下降沿敏感）

图120。 PWM 片段事件选择。当使能输入为高电平时，计数器递增。该使能信号由两个连续级产生。  
首先，四种事件类型中的任意一种（始终开启、引脚 B 为高电平、引脚 B 上升沿、引脚 B 下降沿）都可为分数时钟分频器生成使能脉冲。分频器可降低使能脉冲的频率，然后将其传递至计数器。



请使用各片段 CSR 中的 DIVMODE 字段选择模式。在自由运行模式下，A 和 B 引脚均为输出。在其他任何模式下，B 引脚变为控制计数器操作的输入。非自由运行模式下，CC\_B 信号被忽略。

您可以通过在电平敏感或边沿敏感模式下，使片段运行固定时间来测量输入信号的占空比或频率。由于所采用的边沿检测电路类型，频率测量时，被测信号的低电平周期和高电平周期均须严格大于系统时钟周期。

时钟分频器仍以电平敏感和边沿敏感模式运行。在最大分频 (DIV\_INT 为 0) 情况下，计数器仅在电平敏感模式下的每 256 个高电平输入周期，或边沿敏感模式下的每 256 个边沿时递增一次。这使您能够进行更长时间的测量，尽管分辨率仍保持 16 位。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pwm/measure\\_duty\\_cycle/measure\\_duty\\_cycle.c](https://github.com/raspberrypi/pico-examples/blob/master/pwm/measure_duty_cycle/measure_duty_cycle.c) 第19至37行

```

19 float measure_duty_cycle(uint gpio) {
20 // 仅 PWM 通道 B 的引脚可用作输入。
21 assert(pwm_gpio_to_channel(gpio) == PWM_CHAN_B);
22 uint slice_num = pwm_gpio_to_slice_num(gpio);
23
24 // PWM 通道 B 输入高电平的每 100 个周期计数一次
25 pwm_config cfg = pwm_get_default_config();
26 pwm_config_set_clkdiv_mode(&cfg, PWM_DIV_B_HIGH);
27 pwm_config_set_clkdiv(&cfg, 100);
28 pwm_init(slice_num, &cfg, false);
29 gpio_set_function(gpio, GPIO_FUNC_PWM);
30
31 pwm_set_enabled(slice_num, true);
32 sleep_ms(10);
33 pwm_set_enabled(slice_num, false);
34 float counting_rate = clock_get_hz(clk_sys) / 100;
35 float max_possible_count = counting_rate * 0.01;
36 return pwm_get_counter(slice_num) / max_possible_count;
37 }

```

### 12.5.2.6 配置 PWM 周期

在自由运行模式下，使用以下三个参数控制 PWM 切片输出的周期（以系统时钟周期计）：

- TOP 寄存器，用于控制计数周期的最大值
- CSR\_PH\_CORRECT 位，用于启用相位校正模式
- DIV 寄存器，用于控制时钟分频

切片从 0 计数至 TOP，随后根据 CSR\_PH\_CORRECT 设置，或回绕，或开始反向计数。时钟分频器降低计数速率，最大速度为每个周期计数一次，最小速度为每 256 个周期计数一次。使用以下公式计算时钟周期中的周期数：

$$\text{period} = (\text{TOP} + 1) \times (\text{CSR\_PH\_CORRECT} + 1) \times \left( \text{DIV\_INT} + \frac{\text{DIV\_FRAC}}{16} \right)$$

根据系统时钟频率确定输出频率，请使用以下公式：

$$f_{PWM} = \frac{f_{sys}}{\text{period}} = \frac{f_{sys}}{(\text{TOP} + 1) \times (\text{CSR\_PH\_CORRECT} + 1) \times \left( \text{DIV\_INT} + \frac{\text{DIV\_FRAC}}{16} \right)}$$

将 `DIV_INT` 设置为 0，以将计数速率分频至最大值 256。当 `DIV_INT` 为 0 时，不得设置任何 `DIV_FRAC` 位。

### 12.5.2.7. 中断请求 (IRQ) 与DMA数据请求 (DREQ)

PWM 模块具有两个 IRQ 输出端口。中断状态寄存器 `INTR`、`INTS0`、`INTS1`、`INTE0` 和 `INTE1` 允许软件：

- 控制各切片触发两个 IRQ 之一
- 检查导致 IRQ 触发的切片
- 清除并确认中断

每当计数器回绕（或在相位校正模式下，计数器返回至 0）时，切片将生成中断请求。该操作会在原始中断状态寄存器 `INT_R` 中设置对应于此片段的标志位。如果该片段的中断在 `INTE` 中已启用，则此标志将触发 PWM 模块的 IRQ，同时该标志也会出现在屏蔽中断状态寄存器 `INTS` 中。

要清除标志，请向 `INTR` 写入掩码值。如下 LED 渐变 SDK 示例所示：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/pwm/led\\_fade/pwm\\_led\\_fade.c](https://github.com/raspberrypi/pico-examples/blob/master/pwm/led_fade/pwm_led_fade.c)

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX 许可证标识符: BSD-3-Clause
5 */
6
7 // 在低亮度与高亮度之间渐变 LED。中断处理程序会更新
8 // 计数器每次回绕时 PWM 片段的输出电平。
9
10 #include "pico/stdc.h"
11 #include <stdio.h>
12 #include "pico/time.h"
13 #include "hardware/irq.h"
14 #include "hardware/pwm.h"
15
16 void on_pwm_wrap() {
17 static int fade = 0;
18 static bool going_up = true;
19 // 清除导致中断的标志
20 pwm_clear_irq(pwm_gpio_to_slice_num(PICO_DEFAULT_LED_PIN));
21
22 if (going_up) {
23 ++fade;
24 if (fade > 255) {
25 fade = 255;
26 going_up = false;
27 }
28 } else {
29 --fade;
30 if (fade < 0) {
31 fade = 0;
32 going_up = true;
33 }
34 }
35 }
```

```

33 }
34 }
35 // 对淡入淡出值进行平方处理，使LED亮度呈现更线性的变化
36 // 此范围与wrap值对应
37 pwm_set_gpio_level(PICO_DEFAULT_LED_PIN, fade * fade);
38 }
39
40 int main() {
41 #ifndef PICO_DEFAULT_LED_PIN
42 #warning pwm/led_fade 示例要求开发板配备标准LED
43 #else
44 // 告知LED引脚由PWM负责其数值控制。
45 gpio_set_function(PICO_DEFAULT_LED_PIN, GPIO_FUNC_PWM);
46 // 确定刚连接到LED引脚的切片编号
47 uint slice_num = pwm_gpio_to_slice_num(PICO_DEFAULT_LED_PIN);
48
49 // 将该切片的IRQ输出掩码映射到PWM模块的单一中断线，
50 // 并注册中断处理程序
51 pwm_clear_irq(slice_num);
52 pwm_set_irq_enabled(slice_num, true);
53 irq_set_exclusive_handler(PWM_DEFAULT_IRQ_NUM(), on_pwm_wrap);
54 irq_set_enabled(PWM_DEFAULT_IRQ_NUM(), true);
55
56 // 获取该切片配置的合理默认值。默认情况下，
57 // 计数器允许在其最大范围内循环（0至 $2^{16}-1$ ）
58 pwm_config config = pwm_get_default_config();
59 // 设置分频，降低计数器时钟至 sysclock 除以该值
60 pwm_config_set_clkdiv(&config, 4.f);
61 // 将配置加载至 PWM 分片，并启动运行。
62 pwm_init(slice_num, &config, true);
63
64 // 从此处起，所有操作均在 PWM 中断处理程序中进行，因此我们可以
65 // 无所事事
66 while (1)
67 tight_loop_contents();
68 #endif
69 }

```

该方案允许多个分片同时产生中断。系统中断处理程序确定最近中断的分片，并对其作出适当处理。通常，这意味着重新加载这些分片的 TOP 或 CC 寄存器，但 PWM 模块也可作为非 PWM 目的的定期中断请求源。

同一脉冲既设置了 INTR 中断标志，也作为 RP2350 系统 DMA 的一周期数据请求。每当 DMA 检测到 DREQ 信号被置位时，会尽快向其预设位置传输一笔数据。结合 CC 和 TOP 的双缓冲特性，DMA 能够以每个计数周期一次传输的速率高效地将数据流传输至 PWM 切片。或者，PWM 切片可以用作 DMA 传输至其他内存映射硬件的节奏计时器。

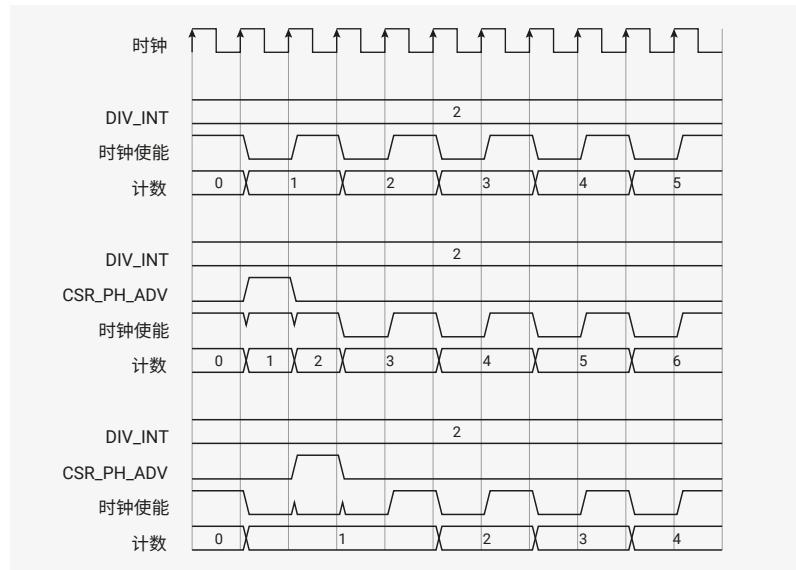
### 12.5.2.8. 动态相位调整

对于某些应用，必须控制两个不同切片上的 PWM 输出之间的相位关系。

全局使能寄存器 EN 包含每个切片的 CSR\_EN 标志的别名。可使用此寄存器同时启动和停止多个切片。如果两个具有相同输出频率的切片同时启动，它们将完美同步运行，其固定相位关系由初始计数值决定。

寄存器字段 CSR\_PH\_ADV 和 CSR\_PH\_RET 可在切片运行时将其输出相位提前或延迟一个计数。其方式是通过向时钟使能信号（时钟分频器输出）插入或删除脉冲，如图 121 所示。

图121。由时钟分频器输出的时钟使能信号控制计数速率。相位提前在时钟使能为低的周期内强制使其为高，导致计数器向前跳过一次计数。相位滞后在时钟使能为高时强制使其为低，使计数器回退一次计数。



计数器计数速度不能超过每个周期一次，因此 `PH_ADV` 要求 `DIV_INT > 1` 或 `DIV_FRAC > 0`。同样，如果在时钟使能永久为低时断言 `PH_RET`，计数器不会开始向后计数。

要使相位提前或滞后一个计数，软件需向 `PH_ADV` 或 `PH_RET` 写入1。一旦插入或删除了一个使能脉冲，`PH_ADV` 或 `PH_RET` 寄存器位将自动恢复为0。软件可轮询 `CSR` 直至该过程完成。`PH_ADV` 始终在下一个可用间隙中插入一个脉冲；`PH_RET` 始终删除下一个可用脉冲。

### 12.5.3. 寄存器列表

PWM寄存器的起始基址为 `0x400a8000`（在SDK中定义为 `PWM_BASE`）。

表1131。  
PWM寄存器列表

偏移量	名称	说明
0x000	<code>CH0_CSR</code>	控制与状态寄存器
0x004	<code>CH0_DIV</code>	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x008	<code>CH0_CTR</code>	对PWM计数器的直接访问
0x00c	<code>CH0_CC</code>	计数器比较值
0x010	<code>CH0_TOP</code>	计数器回绕值
0x014	<code>CH1_CSR</code>	控制与状态寄存器
0x018	<code>CH1_DIV</code>	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x01c	<code>CH1_CTR</code>	对PWM计数器的直接访问
0x020	<code>CH1_CC</code>	计数器比较值
0x024	<code>CH1_TOP</code>	计数器回绕值
0x028	<code>CH2_CSR</code>	控制与状态寄存器
0x02c	<code>CH2_DIV</code>	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。

偏移量	名称	说明
0x030	CH2_CTR	对PWM计数器的直接访问
0x034	CH2_CC	计数器比较值
0x038	CH2_TOP	计数器回绕值
0x03c	CH3_CSR	控制与状态寄存器
0x040	CH3_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x044	CH3_CTR	对PWM计数器的直接访问
0x048	CH3_CC	计数器比较值
0x04c	CH3_TOP	计数器回绕值
0x050	CH4_CSR	控制与状态寄存器
0x054	CH4_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x058	CH4_CTR	对PWM计数器的直接访问
0x05c	CH4_CC	计数器比较值
0x060	CH4_TOP	计数器回绕值
0x064	CH5_CSR	控制与状态寄存器
0x068	CH5_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x06c	CH5_CTR	对PWM计数器的直接访问
0x070	CH5_CC	计数器比较值
0x074	CH5_TOP	计数器回绕值
0x078	CH6_CSR	控制与状态寄存器
0x07c	CH6_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x080	CH6_CTR	对PWM计数器的直接访问
0x084	CH6_CC	计数器比较值
0x088	CH6_TOP	计数器回绕值
0x08c	CH7_CSR	控制与状态寄存器
0x090	CH7_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x094	CH7_CTR	对PWM计数器的直接访问
0x098	CH7_CC	计数器比较值
0x09c	CH7_TOP	计数器回绕值
0x0a0	CH8_CSR	控制与状态寄存器

偏移量	名称	说明
0x0a4	CH8_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x0a8	CH8_CTR	对PWM计数器的直接访问
0x0ac	CH8_CC	计数器比较值
0x0b0	CH8_TOP	计数器回绕值
0x0b4	CH9_CSR	控制与状态寄存器
0x0b8	CH9_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x0bc	CH9_CTR	对PWM计数器的直接访问
0x0c0	CH9_CC	计数器比较值
0x0c4	CH9_TOP	计数器回绕值
0x0c8	CH10_CSR	控制与状态寄存器
0x0cc	CH10_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x0d0	CH10_CTR	对PWM计数器的直接访问
0x0d4	CH10_CC	计数器比较值
0x0d8	CH10_TOP	计数器回绕值
0x0dc	CH11_CSR	控制与状态寄存器
0x0e0	CH11_DIV	INT与FRAC共同构成固定点小数。 计数频率为系统时钟频率除以该数值。 分数除法采用简单的一阶Σ-Δ调制。
0x0e4	CH11_CTR	对PWM计数器的直接访问
0x0e8	CH11_CC	计数器比较值
0x0ec	CH11_TOP	计数器回绕值
0x0f0	EN	此寄存器为所有通道的CSR_EN位的别名。 写入此寄存器可同时启用或禁用多个通道，实现完全同步运行。 每个通道仅有一个物理EN寄存器位， 可通过此处或CHx_CSR进行访问。
0x0f4	中断	原始中断
0x0f8	IRQ0_INTE	irq0 中断使能
0x0fc	IRQ0_INTF	irq0 中断强制
0x100	IRQ0_INTS	irq0 掩码及强制后的中断状态
0x104	IRQ1_INTE	用于 irq1 的中断使能
0x108	IRQ1_INTF	用于 irq1 的中断强制
0x10c	IRQ1_INTS	irq1 掩码和强制后的中断状态

## PWM: CH0\_CSR, CH1\_CSR, ..., CH10\_CSR, CH11\_CSR寄存器

偏移地址: 0x000, 0x014, ..., 0x0c8, 0x0dc

### 描述

控制与状态寄存器

表 1132。CH0\_CSR, CH1\_CSR, ..., CH10\_CSR, CH11\_CSR 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>PH_ADV</b> : 在计数器运行时, 将相位提前1个计数。 自动清零。写入1, 并轮询直至为低。计数器必须以低于全速运行 (div_int + div_frac / 16 > 1)。	SC	0x0
6	<b>PH_RET</b> : 在计数器运行时, 将相位延迟1个计数。 自动清零。写入1, 并轮询直至为低。计数器必须正在运行。	SC	0x0
5:4	<b>DIVMODE</b>	读写	0x0
	枚举值:		
	0x0 → DIV: 以分数分频器确定的速率自由运行计数。		
	0x1 → LEVEL: 分数分频器操作由PWM B引脚控制。		
	0x2 → RISE: 计数器随PWM B引脚的上升沿递增。		
	0x3 → FALL: 计数器随PWM B引脚的下降沿递增。		
3	<b>B_INV</b> : 反转输出B	读写	0x0
2	<b>A_INV</b> : 反转输出A	读写	0x0
1	<b>PH_CORRECT</b> : 1: 启用相位校正调制; 0: 后沿	读写	0x0
0	<b>EN</b> : 使能PWM通道。	读写	0x0

## PWM: CH0\_DIV、CH1\_DIV、...、CH10\_DIV、CH11\_DIV寄存器

偏移量: 0x004, 0x018, ..., 0x0cc, 0x0e0

### 描述

INT与FRAC共同构成固定点小数。  
计数频率为系统时钟频率除以该数值。  
分数除法采用简单的一阶Σ-Δ调制。

表1133 CH0\_DIV、CH1\_DIV、...、CH10\_DIV、CH11\_DIV寄存器

位	描述	类型	复位
31:12	保留。	-	-
11:4	<b>中断</b>	读写	0x01
3:0	<b>FRAC</b>	读写	0x0

## PWM: CH0\_CTR、CH1\_CTR、...、CH10\_CTR、CH11\_CTR寄存器

偏移量: 0x008, 0x01c, ..., 0x0d0, 0x0e4

表1134 CH0\_CTR、CH1\_CTR、...、CH10\_CTR、CH11\_CTR寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	对PWM计数器的直接访问	读写	0x0000

## PWM: CH0\_CC、CH1\_CC、...、CH10\_CC、CH11\_CC 寄存器

偏移量: 0x00c, 0x020, ..., 0xd4, 0xe8

### 描述

计数器比较值

表 1135 CH0\_CC、CH1\_CC、...、CH10\_CC、CH11\_CC 寄存器

位	描述	类型	复位
31:16	B	读写	0x0000
15:0	A	读写	0x0000

## PWM: CH0\_TOP、CH1\_TOP、...、CH10\_TOP、CH11\_TOP 寄存器

偏移量: 0x010, 0x024, ..., 0xd8, 0xec

表 1136 CH0\_TOP、CH1\_TOP、...、CH10\_TOP、CH11\_TOP 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	计数器回绕值	读写	0xffff

## PWM: EN 寄存器

偏移量: 0x0f0

### 描述

此寄存器为所有通道的CSR\_EN位的别名。

写入此寄存器可同时启用或禁用多个通道，实现完全同步运行

。

每个通道仅有一个物理EN寄存器位，

可通过此处或CHx\_CSR进行访问。

表 1137 EN 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	CH11	读写	0x0
10	CH10	读写	0x0
9	CH9	读写	0x0
8	CH8	读写	0x0
7	CH7	读写	0x0
6	CH6	读写	0x0
5	CH5	读写	0x0
4	CH4	读写	0x0
3	CH3	读写	0x0
2	CH2	读写	0x0
1	CH1	读写	0x0

位	描述	类型	复位
0	<b>CH0</b>	读写	0x0

## PWM：INTR寄存器

偏移: 0x0f4

### 描述

原始中断

表 1138. INTR 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	WC	0x0
10	<b>CH10</b>	WC	0x0
9	<b>CH9</b>	WC	0x0
8	<b>CH8</b>	WC	0x0
7	<b>CH7</b>	WC	0x0
6	<b>CH6</b>	WC	0x0
5	<b>CH5</b>	WC	0x0
4	<b>CH4</b>	WC	0x0
3	<b>CH3</b>	WC	0x0
2	<b>CH2</b>	WC	0x0
1	<b>CH1</b>	WC	0x0
0	<b>CH0</b>	WC	0x0

## PWM：IRQ0\_INTE寄存器

偏移量: 0x0f8

### 描述

irq0 中断使能

表 1139。  
IRQ0\_INTE 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	读写	0x0
10	<b>CH10</b>	读写	0x0
9	<b>CH9</b>	读写	0x0
8	<b>CH8</b>	读写	0x0
7	<b>CH7</b>	读写	0x0
6	<b>CH6</b>	读写	0x0
5	<b>CH5</b>	读写	0x0
4	<b>CH4</b>	读写	0x0
3	<b>CH3</b>	读写	0x0
2	<b>CH2</b>	读写	0x0

位	描述	类型	复位
1	<b>CH1</b>	读写	0x0
0	<b>CH0</b>	读写	0x0

## PWM：IRQ0\_INTF寄存器

偏移量: 0x0fc

### 描述

irq0 中断强制

表1140。  
IRQ0\_INTF 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	读写	0x0
10	<b>CH10</b>	读写	0x0
9	<b>CH9</b>	读写	0x0
8	<b>CH8</b>	读写	0x0
7	<b>CH7</b>	读写	0x0
6	<b>CH6</b>	读写	0x0
5	<b>CH5</b>	读写	0x0
4	<b>CH4</b>	读写	0x0
3	<b>CH3</b>	读写	0x0
2	<b>CH2</b>	读写	0x0
1	<b>CH1</b>	读写	0x0
0	<b>CH0</b>	读写	0x0

## PWM：IRQ0\_INTS寄存器

偏移: 0x100

### 描述

irq0 掩码及强制后的中断状态

表1141。  
IRQ0\_INTS 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	只读	0x0
10	<b>CH10</b>	只读	0x0
9	<b>CH9</b>	只读	0x0
8	<b>CH8</b>	只读	0x0
7	<b>CH7</b>	只读	0x0
6	<b>CH6</b>	只读	0x0
5	<b>CH5</b>	只读	0x0
4	<b>CH4</b>	只读	0x0
3	<b>CH3</b>	只读	0x0

位	描述	类型	复位
2	<b>CH2</b>	只读	0x0
1	<b>CH1</b>	只读	0x0
0	<b>CH0</b>	只读	0x0

## PWM：IRQ1\_INTE寄存器

偏移量：0x104

### 描述

用于 irq1 的中断使能

表1142。  
IRQ1\_INTE 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	读写	0x0
10	<b>CH10</b>	读写	0x0
9	<b>CH9</b>	读写	0x0
8	<b>CH8</b>	读写	0x0
7	<b>CH7</b>	读写	0x0
6	<b>CH6</b>	读写	0x0
5	<b>CH5</b>	读写	0x0
4	<b>CH4</b>	读写	0x0
3	<b>CH3</b>	读写	0x0
2	<b>CH2</b>	读写	0x0
1	<b>CH1</b>	读写	0x0
0	<b>CH0</b>	读写	0x0

## PWM：IRQ1\_INTF寄存器

偏移量：0x108

### 描述

用于 irq1 的中断强制

表1143。  
IRQ1\_INTF 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	读写	0x0
10	<b>CH10</b>	读写	0x0
9	<b>CH9</b>	读写	0x0
8	<b>CH8</b>	读写	0x0
7	<b>CH7</b>	读写	0x0
6	<b>CH6</b>	读写	0x0
5	<b>CH5</b>	读写	0x0
4	<b>CH4</b>	读写	0x0

位	描述	类型	复位
3	<b>CH3</b>	读写	0x0
2	<b>CH2</b>	读写	0x0
1	<b>CH1</b>	读写	0x0
0	<b>CH0</b>	读写	0x0

## PWM：IRQ1\_INTS寄存器

偏移：0x10c

### 描述

irq1 掩码和强制后的中断状态

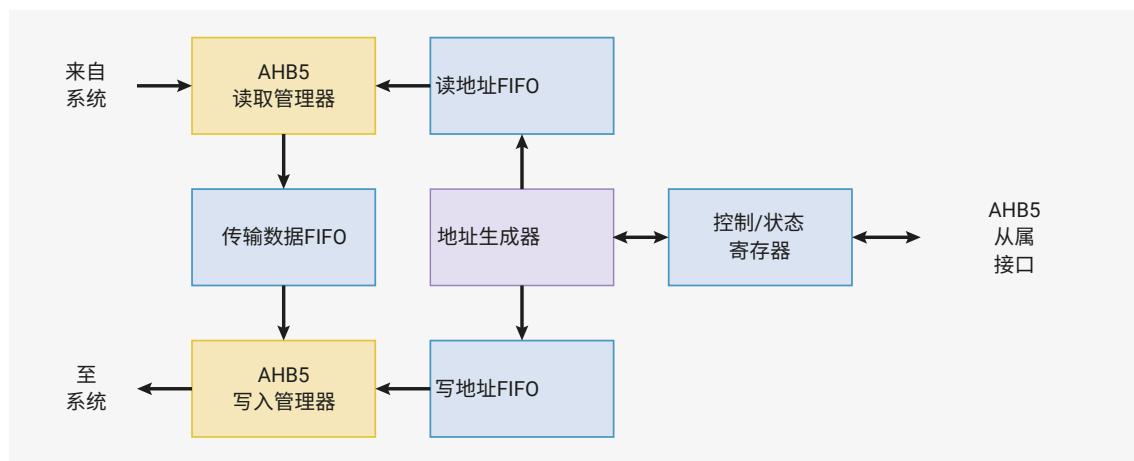
表1144。  
IRQ1\_INTS 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>CH11</b>	只读	0x0
10	<b>CH10</b>	只读	0x0
9	<b>CH9</b>	只读	0x0
8	<b>CH8</b>	只读	0x0
7	<b>CH7</b>	只读	0x0
6	<b>CH6</b>	只读	0x0
5	<b>CH5</b>	只读	0x0
4	<b>CH4</b>	只读	0x0
3	<b>CH3</b>	只读	0x0
2	<b>CH2</b>	只读	0x0
1	<b>CH1</b>	只读	0x0
0	<b>CH0</b>	只读	0x0

## 12.6. DMA

RP2350直接内存访问（DMA）控制器代表处理器执行批量数据传输。这使处理器得以专注于其他任务或进入低功耗睡眠状态。DMA双总线管理端口每个周期可同时发起一次读取和一次写入访问。因此，其数据吞吐量远超单个RP2350处理器。

图122。 DMA  
架构概述。  
读管理器可在每个时钟周期从某个地址读取数据。同样，写管理器可向另一个地址写入数据。地址生成器产生匹配的读写地址对，管理器通过地址FIFO读取这些地址。DMA最多可同时运行16个传输序列，并通过控制与状态寄存器由软件监控。



DMA每个时钟周期可执行一次读访问和一次写访问，数据宽度最高为32位。共有16个独立通道，每个通道监督以下场景之一的总线传输序列：

#### 内存至外设

当外设需要更多传输数据时，会向DMA发出信号。DMA从RAM或闪存中的数组读取数据，并写入外设的数据FIFO。

#### 外设至内存

当外设已接收数据时，会向DMA发出信号。DMA从外设的数据FIFO读取数据，并写入RAM中的数组。

#### 内存到内存

DMA在两个RAM缓冲区之间尽可能快速地传输数据。

每个通道均设有专属控制和状态寄存器（CSR），软件可通过该寄存器对通道进行编程及监控其状态。当多个通道同时激活时，DMA会在这些通道之间均分带宽，并对所有当前请求数据传输的通道采用循环分配策略。

传输大小可为32位、16位或8位。每个通道仅配置一次：源传输大小与目标传输大小须保持一致。DMA对窄写操作执行字节通道复制，因此字节数据在数据总线的所有4个字节中均可用，半字数据亦在两个半字内均可用。

通道可通过多种组合方式实现更加复杂的行为及更高的自主性。例如，一个通道可配置另一个通道，通过内存中一系列控制块载入配置信息，当第二个通道需重新配置时，可通过[CHAIN\\_TO](#)选项回调第一个通道。

使DMA更加自主意味着处理器监督需求显著减少：整体上，这使系统能够同时处理更多任务，或降低功耗。

### 12.6.1. 与 RP2040 的变化

新增以下功能：

- 将DMA通道数量从12增加至16。
- 将共享中断（IRQ）输出数量从2增加至4。
- 通道可通过SECCFG\_CH0至SECCFG\_CH15分配至安全域。
- DMA现利用内置内存保护单元（第12.6.6.3节）过滤总线访问。
- 中断可通过SECCFG\_IRQ0至SECCFG\_IRQ3分配至安全域。
- 节奏计时器和CRC嗅探器可通过SECCFG\_MISC寄存器分配至安全域。
- [TRANS\\_COUNT](#) ([CHO\\_TRANS\\_COUNT](#)) 的最高四位重新定义为[MODE](#)字段，用于定义当[TRANS\\_COUNT](#)达到零时的操作：

- 此向后不兼容的更改将单序列最大传输次数从 $2^{32}-1$ 减少至 $2^{28}-1$ 。
- 模式 **0x0** 的行为与 RP2040 一致，因此无需修改执行每次传输少于 2.56 亿次的软件。
- 模式 **0x1**，“触发自身”，允许通道在完成传输序列后自动重启，除了通常的序列结束动作（例如触发中断或激发其他通道）之外。例如，可利用此模式通过流式环形缓冲区传输实现周期性中断。
- 模式 **0xf**，“无限”，允许通道永久运行：**TRANS\_COUNT** 不递减。
- 新增加的 CH0\_CTRL\_TRIG.INCR\_READ\_REV 和 CH0\_CTRL\_TRIG.INCR\_WRITE\_REV 字段允许地址递减，或按 2 递增。
  - 为容纳新字段，**CTRL** 寄存器中部分现有字段（如 CH0\_CTRL\_TRIG.BUSY）已重新调整位置。

部分现有行为已被优化：

- 针对地址环绕和非递增传输，调节从 **WRITE\_ADDR** 和 **READ\_ADDR** 读取值以反映在途传输数量的逻辑已被禁用（RP2040-E12勘误）。
- 您现在可以轮询 **ABORT** 寄存器，以等待被中止通道的完成（RP2040-E13缺陷修正）。
- DMA 完成操作如 **CHAIN\_TO** 现已严格按照最后写入完成的顺序执行，因此对写入其寄存器的通道执行 **CHAIN\_TO** 是一个明确规定操作。
  - 这使得能够使用不包含四个触发寄存器别名中的任意一个的控制块。
  - 此前，通道被视为在其最后一次写入数据阶段的第一个周期完成。现今，通道被视为在其最后一次写入数据阶段的最后一个周期完成。通常这两个周期相同，但当 DMA 遇到写数据阶段的总线阻塞时，完成周期可能会延后。
- 此前，DMA 的内部仲裁逻辑在完成一轮活动高优先级通道（CH0\_CTRL\_TRIG.HIGH\_PRIORITY）后插入一个空闲周期，即使无任何活动低优先级请求。这导致 DMA 在轻负载情况下吞吐量降低。该空闲周期已被移除，消除吞吐量损失。
- 中断请求（IRQ）断言延迟已减少一个周期。

## 12.6.2. 配置通道

每个通道均设有四个控制/状态寄存器：

- **READ\_ADDR** (CH0\_READ\_ADDR) 为下一次读取的内存地址。
- **WRITE\_ADDR** (CH0\_WRITE\_ADDR) 为下一次写入的内存地址。
- **TRANS\_COUNT** (CH0\_TRANS\_COUNT) 显示当前传输序列中剩余的传输次数，并设置下一传输序列的传输次数（详见第 12.6.2.2 节）。
- **CTRL** (CH0\_CTRL\_TRIG) 配置通道行为的所有其他方面，启用或禁用通道，并提供完成状态。

为直接指示 DMA 通道执行数据传输，软件需要写入这四个寄存器，然后触发该通道（详见第 12.6.3 节）。为增强 DMA 的自主性，您还可以编程使一个 DMA 通道写入另一个通道的配置寄存器，从而预先排队多个传输序列。

这四个均为活动寄存器；它们在通道执行过程中持续更新状态。

### 12.6.2.1 读写地址

**READ\_ADDR** 和 **WRITE\_ADDR** 分别存储通道下一次读取和写入的地址。这些寄存器在每次读写访问后会自动更新，按需递增至下一个读写地址。递增大小依据以下因素变化：

- 传输大小：依据CH0\_CTRL\_TRIG.DATA\_SIZE的1、2或4字节总线访问
- 各地址寄存器的递增使能标志：CH0\_CTRL\_TRIG.INCR\_READ和CH0\_CTRL\_TRIG.INCR\_WRITE
- 递增方向：CH0\_CTRL\_TRIG.INCR\_READ\_REV和CH0\_CTRL\_TRIG.INCR\_WRITE\_REV

软件通常应在每次新传输序列开始时，重新编程这些寄存器的起始地址。

若 `READ_ADDR` 和 `WRITE_ADDR` 未被重新编程，DMA 将使用当前值作为下一次传输的起始地址。例如：

- 如果地址不递增（例如，该地址为外围设备 FIFO 的地址），且下一传输序列仍针对该相同地址进行，则无需再次写入该寄存器。
- 当向内存中连续的一系列缓冲区传输数据时（例如，分散与聚集操作），地址寄存器在传输完成后将自动递增至下一缓冲区的起始地址。

通过不为每个传输序列编程所有四个CSR，软件能够使用更简短的中断处理程序，并在采用通道链时实现更紧凑的控制块格式（详见第12.6.3.1节寄存器别名及第12.6.3.2节链式结构）。

#### 12.6.2.1.1. 地址对齐

`READ_ADDR` 和 `WRITE_ADDR` 必须与传输大小对齐，传输大小由 `CH0_CTRL_TRIG.DATA_SIZE` 指定。对于32位传输，地址必须为4的倍数；对于16位传输，地址必须为2的倍数。

软件负责正确对齐写入 `READ_ADDR` 和 `WRITE_ADDR` 的地址；DMA 不执行对齐检查。

如果软件最初写入正确对齐的地址，则在整个传输序列中该地址将保持正确对齐，因为 DMA 始终以传输大小的整数倍递增 `READ_ADDR` 和 `WRITE_ADDR`。具体而言，递增量为传输大小乘以-1、0、1或2，具体取决于 `CH0_CTRL_TRIG.INCR_READ`、`CH0_CTRL_TRIG.INCR_WRITE`、`CH0_CTRL_TRIG.INCR_READ_REV` 和 `CH0_CTRL_TRIG.INCR_WRITE_REV` 的取值。

DMA MPU 及系统级总线安全过滤器对每个周期传输的所有字节的最低字节地址（即当前 `READ_ADDR` / `WRITE_ADDR` 的值）进行保护检查。RP2350 内存硬件确保未对齐的总线访问不会导致数据从保护边界的另一侧被读取或写入。这表明未对齐访问不能用来违反内存保护模型。除此之外，未对齐访问的结果是不确定的。

#### 12.6.2.2. 传输计数

读取 `TRANS_COUNT` (`CH0_TRANS_COUNT`) 返回当前传输序列中剩余的传输次数。

该数值随着通道的进度持续更新。写入 `TRANS_COUNT` 可设置下一次传输序列的长度。一次序列中最多可执行  $2^{28-1}$  次传输（即 `0xffffffff`，约 2.56 亿次）。

每当通道启动新的传输序列时，最近写入的 `TRANS_COUNT` 值会复制到当前的传输计数器，然后该计数器会随着新传输序列的进行再次递减。用于调试的 `DBG_TCR` (`TRANS_COUNT` 重载值) 寄存器显示各通道最后写入的 `TRANS_COUNT` 值。

如果通道在未对 `TRANS_COUNT` 进行写入的情况下多次触发，则每次执行的传输次数相同。例如，当链式连接时，一个通道可能会将固定大小的控制块加载到另一个通道的CSR寄存器中。软件仅需编程一次 `TRANS_COUNT`，随后每次都会自动重载。

或者，可以在开始每次传输序列之前写入新的 `TRANS_COUNT` 值。如果 `TRANS_COUNT` 是通道触发器（参见第12.6.3.1节），通道会立即启动，并采用新写入的数值，而非重载寄存器中当前的数值。

### ⓘ 注意

`TRANS_COUNT`表示将执行的传输次数。传输的字节总数等于`TRANS_COUNT`与每次传输字节大小（由`CTRL.DATA_SIZE`给出）的乘积。

#### 12.6.2.1. 计数模式

`TRANS_COUNT`的最高四位包含`MODE`字段（`CH0_TRANS_COUNT.MODE`），该字段会修改`TRANS_COUNT`的计数行为。模式`0x0`为默认模式：`TRANS_COUNT`每完成一次总线传输递减一次，当`TRANS_COUNT`归零且所有正在进行的传输完成后，通道停止工作。选择值`0x0`是为了向后兼容RP2040软件，该软件期望`TRANS_COUNT`寄存器包含32位计数，而非4位模式和28位计数。有限传输次数大于228的用例极少，因此最高四位已重新分配，用于支持无限次传输。

模式`0x1`，即`TRIGGER_SELF`，行为与模式`0x0`相同，但完成后通道不会停止，而是立即自我触发。这相当于通过其他机制执行的触发（参见第12.6.3节）：

重新加载`TRANS_COUNT`，通道从当前`READ_ADDR`和`WRITE_ADDR`地址继续运行。若`CTRL.IRQ QUIET`未设置，仍会产生完成中断，并且仍会执行指定的`CHAIN_TO`操作。此模式的主要用途是通过SRAM环形缓冲区进行流式传输，在其中需定期执行某些操作，例如在音频缓冲区使用至一半时请求处理器重新填充音频缓冲区。

模式`0xf`（`ENDLESS`）禁用`TRANS_COUNT`的递减。这意味着通道通常会无限期运行且不中断，但以模式`0x0`与计数`0x0`触发通道将导致通道立即停止。

所有其他数值均为保留以备将来使用，其效果未作说明。

#### 12.6.2.3 控制/状态

`CTRL`寄存器（`CH0_CTRL_TRIG`）包含比其他三个寄存器更多且更小的字段。除其他内容外，`CTRL`用于：

- 通过`DATA_SIZE`字段配置该通道数据传输的大小。读取操作的大小始终与写入操作相同。
- 通过`INCR_READ`、`INCR_READ_REV`、`INCR_WRITE`、`INCR_WRITE_REV`、`RING_SEL`和`RING_SIZE`字段，配置`READ_ADDR`和`WRITE_ADDR`在每次读写后的递增方式及是否递增。支持环形传输，即地址指针在2的幂边界处回绕。
- 通过`CHAIN_TO`字段选择另一个通道（或无）以在此通道完成时触发。
- 通过`TREQ_SEL`字段选择外设数据请求（DREQ）信号，以调节该通道的传输。
- 通过`BUSY`标志检测通道是否空闲。
- 通过`READ_ERROR`和`WRITE_ERROR`标志，或通过`AHB_ERROR`标志的综合错误状态，判断通道是否发生总线错误。

#### 12.6.3. 触发通道

通道配置完成后，须触发该通道。此操作指示其开始调度总线访问，方式为由外设数据请求信号（DREQ）节奏控制，或尽可能高速执行。以下事件可触发通道：

- 对通道触发寄存器进行写入。
- 另一个`CHAIN_TO`指向本通道的通道完成。
- 对`MULTI_CHAN_TRIGGER`寄存器进行写入（可同时触发多个通道）。

各触发机制适用于不同的使用场景。例如，当

在中断服务例程中配置和启动通道时，触发寄存器简单且高效，因为通道由最后一次配置写操作触发。`CHAIN_TO` 允许一个通道回调到另一个通道，后者随后可以重新配置第一个通道。`MULTI_CHAN_TRIGGER` 允许软件简单启动一个通道，而无需修改其任何配置寄存器。

触发时，通道会设置其 `CTRL.BUSY` 标志，以指示该通道正在积极调度传输。该标志保持设置状态，直到传输计数归零，或通过 `CHAN_ABORT` 寄存器（第12.6.8.3节）中止该通道。

当通道已运行 (`BUSY = 1`) 时，会忽略额外的触发信号。已禁用的通道 (`CTRL.EN` 位清零) 同样会忽略触发信号。

### 12.6.3.1. 别名与触发

表1145。控制寄存器别名每个通道包含四个控制/状态寄存器。每个寄存器可通过多个不同地址进行访问。在每个自然对齐的四寄存器组内，所有四个寄存器均出现，但顺序不同。

偏移量	+0x0	+0x4	+0x8	+0xc (触发)
<code>0x00</code> (别名 0)	<code>READ_ADDR</code>	<code>WRITE_ADDR</code>	<code>TRANS_COUNT</code>	<code>CTRL_TRIG</code>
<code>0x10</code> (别名 1)	<code>CTRL</code>	<code>READ_ADDR</code>	<code>WRITE_ADDR</code>	<code>TRANS_COUNT_TRIG</code>
<code>0x20</code> (别名 2)	<code>CTRL</code>	<code>TRANS_COUNT</code>	<code>READ_ADDR</code>	<code>WRITE_ADDR_TRIG</code>
<code>0x30</code> (别名 3)	<code>CTRL</code>	<code>WRITE_ADDR</code>	<code>TRANS_COUNT</code>	<code>READ_ADD_TRIG</code>

这四个 CSR 在内存中被多次别名。每个别名均映射相同的四个物理寄存器，但顺序各异。每个别名中最后一个寄存器（偏移 +0xc，已高亮）为触发寄存器。向触发寄存器写入数据将启动该通道。

通常仅使用别名 0，别名 1 至 3 可予以忽略。配置并启动通道时，请依次写入 `READ_ADDR`、`WRITE_ADDR`、`TRANS_COUNT`，最后写入 `CTRL`。由于 `CTRL` 是别名 0 中的触发寄存器，因此它启动通道。

其他别名在使用一个通道配置另一个通道时，允许更紧凑的控制块列表，并能在中断处理程序中实现更高效的重新配置与启动：

- 每个CSR都是某一别名中的触发寄存器：
  - 在将固定大小的缓冲区聚集到外围设备时，DMA通道可通过仅写入 `READ_ADDR_TRIG` 来配置并启动。
  - 在将数据从外围设备分散至固定大小缓冲区时，通道可通过仅写入 `WRITE_ADDR_TRIG` 来配置并启动。
- 寄存器的有效组合表现为包含触发寄存器的自然对齐元组。结合通道链与地址环绕，这些组合实现了压缩控制块格式，例如：
  - (`WRITE_ADDR`, `TRANS_COUNT_TRIG`) 用于外围设备的分散操作
  - ((`TRANS_COUNT`,`READ_ADDR_TRIG`) 用于外围设备的聚集操作，或对缓冲区列表进行CRC计算
  - ((`READ_ADDR`,`WRITE_ADDR_TRIG`) 用于操作内存中的固定大小缓冲区

触发寄存器在以下情况下不会启动通道：

- 通过 `CTRL.EN` 禁用通道（若触发类型为 `CTRL`，则使用刚写入的 `EN` 值，而非 `CTRL` 寄存器中当前的值）
- 通道已处于运行状态
- 向触发寄存器写入值 0（用于结束控制块链，详见空触发器（第12.6.3.3节））
- 总线访问安全级别低于通道安全级别（第12.6.6.1节）

### 12.6.3.2. 链接

当通道完成时，可指定不同通道立即触发。此功能可作为第二通道的回调，用于重新配置并重新启动第一个通道。

通过通道 `CTRL` 寄存器中的 `CHAIN_TO` 字段配置该功能。该4位值选择在当前通道结束后将启动的通道。通道不得链接自身。将 `CHAIN_TO` 设置为通道自身的索引可防止链式连接。

链式触发的行为与其他来源的触发器相同，例如触发寄存器。例如，它们会导致 `TRANS_COUNT` 重新加载，且若目标通道已在运行，则会被忽略。

链式连接的一种应用是通道请求由内存中一系列控制块的另一个通道进行重新配置。通道A被配置为从内存向通道B的控制寄存器（包括触发寄存器）执行环形传输，通道B则配置为在完成每个传输序列后链回通道A。此情况在DMA控制块示例中有明确展示（第12.6.9.2节）。

使用寄存器别名（第12.6.3.1节）可实现DMA控制块的紧凑格式：某些情况下仅需一个字。

链式连接的另一种用途是乒乓配置，其中两个通道相互触发。处理器可以响应通道完成中断，并在每个通道完成后重新配置其参数。然而，已配置好的链式通道会立即开始运行。换言之，通道配置和通道操作是流水线并行的。当使用模式需要多次短传输序列时，此方式能极大提升性能。

第12.6.9节详细介绍了链式触发器在实际应用中的各种可能。

### 12.6.3.3. 空触发与链式中断

如第12.6.3.1节所述，向触发寄存器写入全零不会启动通道。这称为空触发器，具有以下两个目的：

- 通过附加一个全零控制块，实现在控制块数组末尾停止操作。
- 减少使用控制块时所产生的中断数量。

默认情况下，除非在 `INTE0` 至 `INTE3` 中屏蔽了该通道的IRQ，否则每完成一次传输序列，通道都会产生一次中断。中断频率可能过高，尤其是在连续执行一系列控制块时，通常不需要处理器介入。然而，链条末端需要处理器关注。

通道 `CTRL` 寄存器包含一个名为 `IRQ QUIET` 的字段。其默认值为0。当该字段设为1时，通道在接收到空触发时会产生中断，但在正常完成传输序列时不会产生中断。中断由接收到触发信号的通道触发。

## 12.6.4. 数据请求 (DREQ)

外设以其自身速率生成或消耗数据。如果DMA以最大速度传输数据，将导致数据丢失或损坏。DREQ是外设与DMA之间的通信通道，使DMA能够根据外设需求调节传输节奏。

字段 `CTRL.TREQ_SEL`（传输请求）用于选择外部DREQ，亦可用于选择内部节奏定时器之一，或不选择任何TREQ（传输将尽可能快速进行），例如用于内存到内存的传输。

### 12.6.4.1 系统DREQ表

DREQ 编号按照以下方式全局分配至外设 DREQ 通道：

表 1146. DREQs

DREQ	DREQ 通道	DREQ	DREQ 通道	DREQ	DREQ 通道	DREQ	DREQ 通道
0	DREQ_PI00_TX0	14	DREQ_PI01_RX2	28	DREQ_UART0_TX	42	DREQ_PWM_WRAP10
1	DREQ_PI00_TX1	15	DREQ_PI01_RX3	29	DREQ_UART0_RX	43	DREQ_PWM_WRAP11
2	DREQ_PI00_TX2	16	DREQ_PI02_TX0	30	DREQ_UART1_TX	44	DREQ_I2C0_TX
3	DREQ_PI00_TX3	17	DREQ_PI02_TX1	31	DREQ_UART1_RX	45	DREQ_I2C0_RX
4	DREQ_PI00_RX0	18	DREQ_PI02_TX2	32	DREQ_PWM_WRAP0	46	DREQ_I2C1_TX
5	DREQ_PI00_RX1	19	DREQ_PI02_TX3	33	DREQ_PWM_WRAP1	47	DREQ_I2C1_RX
6	DREQ_PI00_RX2	20	DREQ_PI02_RX0	34	DREQ_PWM_WRAP2	48	DREQ_ADC
7	DREQ_PI00_RX3	21	DREQ_PI02_RX1	35	DREQ_PWM_WRAP3	49	DREQ_XIP_STREAM
8	DREQ_PI01_TX0	22	DREQ_PI02_RX2	36	DREQ_PWM_WRAP4	50	DREQ_XIP_QMITX
9	DREQ_PI01_TX1	23	DREQ_PI02_RX3	37	DREQ_PWM_WRAP5	51	DREQ_XIP_QMIRX
10	DREQ_PI01_TX2	24	DREQ_SPI0_TX	38	DREQ_PWM_WRAP6	52	DREQ_HSTX
11	DREQ_PI01_TX3	25	DREQ_SPI0_RX	39	DREQ_PWM_WRAP7	53	DREQ_CORESIGHT
12	DREQ_PI01_RX0	26	DREQ_SPI1_TX	40	DREQ_PWM_WRAP8	54	DREQ_SHA256
13	DREQ_PI01_RX1	27	DREQ_SPI1_RX	41	DREQ_PWM_WRAP9		

#### 12.6.4.2. 基于信用的 DREQ 方案

RP2350 DMA 设计适用于如下系统环境：

- 大型外设数据 FIFO 的面积和功耗成本过高。
- 单个外设的带宽需求可能较大，例如短时间内总线注入率超过 50%。
- 总线延迟较低，但多个管理器可能竞争总线访问权。

此外，DMA 的传输 FIFO 及双管理器端口结构允许对同一外设的多次访问同时进行，以提升吞吐量。因此，DREQ 机制的选用至关重要：

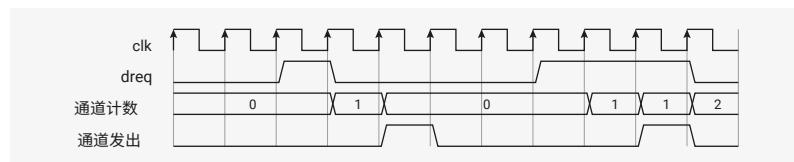
- 传统的“打开水龙头”方法在多个写入积压于 TDF 时可能导致溢出。部分系统通过超额配置外设 FIFO 并将 DR\_EQ 阈值设定在满容量以下解决此问题，尽管会牺牲宝贵的面积和功耗。
- Arm风格的单次和突发握手不允许在当前请求处理中注册额外请求。当FIFO深度较浅时，此限制会影响性能。

RP2350 DMA采用基于信用的DREQ机制。对于每个外设，DMA尝试保持尽可能多的传输同时进行，数量受外设容量限制。这使得在无结构延迟或争用的情况下，通过8级深度外设FIFO实现全总线吞吐量（每时钟1字）成为可能，且无溢出或欠载风险。

每个通道，DMA都会维护一个计数器。每个针对 `dreq`信号的1时钟脉冲都会使该计数器递增。当计数器非零时，通道向DMA内部仲裁器请求传输。当传输发往地址FIFO时计数器递减，此时传输已启动，但可能尚未完成。

该计数器为六位饱和计数器。计数器在达到最大值时忽略递增操作，在为零时忽略递减操作。六位计数器支持计数至RP2350上任意FIFO的深度。

图123。DREQ  
计数



其效果是在外围设备FIFO中可用空间或数据量的基础上限制在途传输的最大数量。在稳态下，此方式保证最大吞吐量，且不会发生溢出或下溢。该方法具备以下注意事项：

- 用户不得访问DMA当前正在服务的FIFO，否则将导致通道与外围设备不同步，可能引发数据损坏或丢失。
- 多个通道不得连接至相同的DREQ。

## 12.6.5. 中断

每个通道均可产生中断；这些中断可通过四个相同的中断使能寄存器INTE0至INTE3按通道分别屏蔽。通道发出中断请求的情况有三种：

- 每个传输序列完成时，若`CTRL.IRQ.QUIET`未启用
- 在接收到空触发时，若启用了`CTRL.IRQ.QUIET`
- 发生读或写总线错误时

掩码中断状态可在 `INTS` 寄存器中查看；每个通道对应一位。通过向 `INTS` 寄存器写入位掩码以清除中断。确认中断的一种惯用做法是先读取 `INTS`，然后将相同的值写回，从而仅清除已启用的中断。

RP2350 DMA 提供四个系统 IRQ，配备独立的屏蔽和状态寄存器（例如 `INTE0` 和 `INTE1`）。任意组合的通道中断请求可路由至每个系统 IRQ，尽管通常软件只将每个通道中断路由至单个系统 IRQ。例如：

- 若某些通道具有特别严格的时序需求，则可在系统中断控制器中赋予其更高优先级。
- 在多处理器系统中，不同通道中断可独立路由至不同内核。
- 当通道分配至混合安全域时，IRQ 亦可分配，以使各安全域的软件能接收来自其自身通道的中断。

出于调试目的，`INTF` 寄存器可强制任意通道中断被激活，从而导致在对应 `INTE` 寄存器中设置了该通道中断使能位的任何系统 IRQ 被断言。

## 12.6.6. 安全性

RP2350 的处理器支持将内存和外设划分为多个安全域。此划分扩展至 DMA，使不同安全上下文能够安全地使用其分配的通道，而不破坏处理器安全模型所规定的任何安全不变量。例如，处于非安全态的 Arm 处理器不得通过 DMA 访问由安全软件拥有的内存或外设。

DMA 定义了四个安全级别，映射至 Arm 或 RISC-V 处理器的安全状态：

- 3: SP (安全且特权)
  - 等同于处于安全且特权态的 Arm 处理器
  - 等同于处于机器模式的 RISC-V 处理器
- 2: SU (安全且非特权)

- 等同于处于安全且普通态的 Arm 处理器
- 1: NSP (非安全且特权)
  - 等同于处于非安全、特权态的 Arm 处理器
  - 等同于处于监督模式的 RISC-V 处理器
- 0: NSU (非安全且非特权)
  - 等同于处于非安全、正常态的 Arm 处理器
  - 等同于处于用户模式的 RISC-V 处理器

为使 DMA 能够以一致方式比较不同安全级别，安全级别被视为有序，顺序为 SP > SU > NSP > NSU。例如，当我们说某通道访问其寄存器所需的最低安全级别为 SU 时，表示 SP 和 SU 可接受，而 NSP 和 NSU 不可接受。通常，每个动作对应的响应安全级别不高于该动作，因此 DMA 不能用于提升访问权限至更高安全级别。

软件将内部 DMA 资源，如通道、中断、节奏定时器及 CRC 嗅探器，分配至四个可能的安全级别之一。这些资源随后仅在该安全级别及以上可访问。频道分配的具体内容详见第12.6.6.1节。

DMA内存保护单元（第12.6.6.3节）定义了访问多达八个可编程地址范围所需的最低安全级别，确保具有特定安全级别的通道无法访问超出其权限的内存区域。

该MPU旨在映射处理器SAU或PMP中配置的SRAM和XIP内存保护边界。除DMA MPU执行的内部过滤外，访问还会根据第10.6.2节中描述的ACCESSCTRL过滤规则，由系统总线进行过滤。

这些功能的结合使DMA能够安全地被运行于不同安全域的软件共享。如不希望如此，可通过ACCESSCTRL DMA寄存器将整个DMA模块完整分配至单一安全域。

### 12.6.6.1. 通道安全分配

通道通过通道SECCFG寄存器（SECCFG\_CH0至SECCFG\_CH15）分配至安全域。

每个通道对应一个寄存器。每个寄存器包含一个2位的安全级别和一个锁定位，该锁定位防止该 SECCFG 寄存器在配置后被更改。复位时，所有通道均被分配为SP安全级别，且该级别为最高。

通道的安全级别定义如下：

- 该通道执行的总线传输的安全级别，将同时与DMA内存保护单元及第10.6.2节所述的ACCESSCTRL总线级过滤器进行校验。
- 读取或写入该通道寄存器所需的最低安全级别；低于该级别的访问将返回总线错误。
- 共享IRQ线上必须定义的最低安全级别，以使该IRQ能够监视此通道的中断（见第12.6.6.2节），或通过该IRQ的寄存器设置/清除该通道中断。
- 通过INTR寄存器清除该通道中断所需的最低总线安全级别。
- 通道能够监视的DREQ信号：分配为NSP或NSU安全级别的通道无法监视仅限安全外设的DREQ（如ACCESSCTRL外设配置所定义）。
- 可以观察到哪些pacing timer TREQ；pacing timer的安全级别由SECCFG\_MISC配置，且不得高于该通道的通道安全级别，方可观察TREQ。
- 该通道是否对CRC嗅探器可见；嗅探器的安全级别由SECCFG\_MISC配置，且不得低于被观察通道的安全级别。
- 该通道可通过 CHAIN\_TO 触发哪些通道；不允许从较低安全级别向较高安全级别的链式触发。

- 通过向MULTI\_CHAN\_TRIGGER写入触发该通道所需的最低总线安全级别。

通道 **SECCFG** 寄存器要求特权写入（SP/NSP），非特权写入（SU/NSU）将导致总线错误。此外，**S**位（安全级别最高有效位）和**LOCK**位仅限SP写入；而**P**位（安全级别最低有效位）仅在**S**位为0时，方可由NSP写入。读取始终被允许：始终可以通过读取通道 **SECCFG** 寄存器查询分配给您的通道。

每个通道 **SECCFG** 寄存器可通过向该寄存器中的 **LOCK**位写入1来手动锁定，且在对该通道的任一控制寄存器（如CH0\_CTRL\_TRIG）成功写入后会自动锁定。此自动锁定避免了通道安全级别在开始传输后变化，或控制寄存器中已写入的安全指针泄露可能引发的竞态条件。通道**SECCFG** 寄存器锁定后即变为只读。**LOCK**位仅能通过对DMA模块进行完全复位来清除。

**SECCFG** 寄存器在锁定前可多次写入，因此无需预先完全确定分配方案：例如，安全Arm软件可在启动非安全软件上下文之前将备用通道设置为NSP，非安全特权软件随后可将不再需要的剩余通道设为NSU，之后返回至非安全普通上下文。

### 12.6.6.2. 中断安全分配

RP2350 DMA具有四条系统级中断请求线（IRQ），每条IRQ可由通道中断的任意组合触发，其组合由中断使能寄存器INT\_E0至INT\_E3中的通道掩码定义。鉴于中断时序可能泄露信息，且恶意操控中断可能导致软件异常，必须严格控制对通道中断标志的访问。

中断安全配置寄存器SECCFG IRQ0至SECCFG IRQ3定义了各中断的安全级别。该安全级别为第12.6.6节中列示的四个安全等级之一。IRQ的安全级别定义了：

- 该IRQ状态寄存器中哪些通道可见；安全级别高于该IRQ的通道将读回零值。
- 是否允许对该IRQ的控制和状态寄存器进行总线访问；低于该IRQ安全级别的总线访问将导致总线错误，且不会对DMA产生任何影响。
- 哪些通道会断言该IRQ；高于该IRQ级别的通道不会导致中断断言，即使相关的INTE位已设置。
- 某通道的中断是否可以通过该IRQ的INTS寄存器清除，或通过该通道的INTF寄存器设置；高于该IRQ安全级别的通道的中断标志无法被设置或清除。

INTR寄存器由所有IRQ共用，因此不遵守任何IRQ的安全级别。相反，其遵循总线访问的安全级别：读取INTR将返回所有处于总线访问安全级别及以下的通道的中断标志（高于该级别的通道读取结果为零），对INTR的写入具有写一清零行为，且仅针对处于总线访问安全级别及以下的通道生效。

### 12.6.6.3. 内存保护单元

DMA内存保护单元（MPU）监控DMA执行的所有读/写传输的地址，并记录源通道的安全级别。MPU预先配置了用户定义的安全地址映射，该映射指定了访问最多八个动态配置区域所需的最低安全级别。

这是第12.6.6节中定义的四个安全级别之一。

未达到其地址最低安全级别的传输在到达系统总线之前将被终止，并向起始通道返回总线错误。这将在通道的 **CTRL** 寄存器中报告为读或写总线错误，具体取决于安全检查失败的是读地址还是写地址。

DMA MPU的预期用途是反映处理器SAU或PMP中SRAM和XIP存储器的安全定义。由于DMA MPU区域数量不足以分配给各个外围设备，因此为此目的提供了ACCESSCTRL总线访问寄存器（第10.6.2节）。

八个MPU区域中的每一个均配置有基址，分别为MPU\_BAR0至MPU\_BAR7，以及限制地址，分别为MPU\_LAR0至MPU\_LAR7。

MPU区域的粒度为32字节，因此基址/限制地址由每个 BAR/LAR寄存器的27个最高有效位（位 31:5）配置。当地址的27个最高有效位大于或等于 BAR地址位且小于或等于 LAR地址位时，地址匹配MPU区域。例如，当MPU\_BAR0和MPU\_LAR0的值均为 0x10000000时，MPU区域0匹配一个从字节地址 0x10000000至 0x1000001f（含）扩展的32字节区域。区域可通过 LAR.E位启用或禁用——若区域被禁用，则不匹配任何地址。

访问每个区域所需的最低安全级别由该区域LAR寄存器最低有效位的S和P位定义。当地址匹配多个区域时，优先适用编号最低的区域。此规则符合RISC-V PMP的平局判定规则，但与Arm SAU的平局判定规则不同，因此在镜像具有重叠区域的SAU映射时务必谨慎。当没有任何MPU区域匹配时，安全级别由全局MPU\_CTRL.S和MPU\_CTRL.P位确定。

MPU配置寄存器（MPU\_CTRL、MPU\_BAR0至MPU\_BAR7及MPU\_LAR0至MPU\_LAR7）不允许非特权访问。在SU和NSU安全级别下的总线访问将产生总线错误且不产生其他影响。

MPU寄存器对NSP访问大多为只读，唯一例外是区域 P位，且仅当对应区域的 S位清零时，该P位可由NSP写入。此项委托特权非安全软件决定非安全区域是否允许NSU访问。

## 12.6.7. 总线错误处理

总线错误是指DMA管理端口之一所报告的错误状态，表明尝试的读写传输因下列原因之一被拒绝：

- DMA MPU禁止源通道在其安全级别访问该地址（见第12.6.3节）。
- 总线结构未能解码该地址；该地址未匹配任何已知内存位置（例如SIO因紧密耦合于处理器，从DMA总线端口不可见）。
- ACCESSCTRL禁止源通道在其特权级别访问所寻址区域（见第10.6.2节）。
- ACCESSCTRL不论特权级别，均禁止DMA访问所寻址区域。
- APB桥在传输超时超过65535周期时返回超时故障（例如访问ADC时 clk\_adc 已停止）。
- 下游总线端口因其他设备特定原因返回错误响应，例如尝试访问安全级别更高的DMA通道配置寄存器（见第12.6.1节）。

### 12.6.7.1. 总线错误响应

遇到总线错误时，DMA将停止违规通道，并通过该通道的CHO\_CTRL\_TRIG.READ\_ERROR及WRITE\_ERROR标志报告该错误。通道停止调度总线访问。

总线错误为异常事件，通常指示DMA或其他系统硬件配置错误。

因此，DMA拒绝重新启动故障通道，直到通过向相关错误标志写入 1清除其错误状态。其他通道不受影响，继续不间断地执行传输序列。

遇到总线错误的通道不会 CHAIN\_TO 其他通道。

总线错误必然会触发通道的中断请求。是否产生系统级IRQ取决于中断使能寄存器INTE0至INTE3中配置的通道屏蔽状态。

### 12.6.7.2. 总线错误后的恢复

若通过 `READ_ERR`/`WRITE_ERR` 报告错误，则在重启通道前，软件必须：

1. 轮询 `BUSY` 信号至低电平，确保该通道所有在执行中的传输已从 DMA 总线流水线中清除。
2. 通过向每个错误标志写入 1 以清除错误标志。

通常，处理器进入中断处理程序并检查错误状态前，`BUSY` 标志早已变为低电平，但当 DMA 访问诸如 XIP 这类高 `SCK` 分频慢速设备且处理器从 `SRAM` 执行时，这些事件可能存在重叠。

`READ_ADDR` 和 `WRITE_ADDR` 包含发生总线错误的大致地址。这有助于程序员理解总线错误发生的原因，并修正软件以避免将来再发生此类错误。

由于 DMA 并行执行读写操作，通道可能同时遇到读错误和写错误，在这种情况下，DMA 会同时设置 `READ_ERR` 和 `WRITE_ERR`。您必须同时清除两者。

### 12.6.7.3. 停止时序

DMA会在发生总线错误后尽快停止该通道。这将抑制后续的读写操作。由于访问总线的请求被屏蔽，访问总线对系统没有副作用。由于 DMA 的流水线和缓冲机制，时序关系较为复杂。DMA 对于来自同一通道的传输提供以下顺序保证：

- 读错误 → 读操作抑制：任何计划在故障读操作之后执行的读操作将被抑制，但仍可最多递增 `READ_ADDR` 两次
- 写入错误 → 写入抑制：任何计划在出错写入之后进行的写入将被抑制，但仍可最多增加 `WRITE_ADDR` 四次
- 读取错误 → 写入抑制：
  - 任何与出错读取配对的写入将被抑制，但会增加 `WRITE_ADDR`
  - 紧随与出错读取配对的首次写入之后的任何写入将被抑制，但最多可增加 `WRITE_ADDR` 三次
  - 紧接与出错读取配对的首次写入之前的最多三次写入可被抑制，但会增加 `WRITE_ADDR`
- 写入错误 → 读取抑制：
  - 与首次出错写入之前的写入配对的读取不会被抑制，且会增加 `READ_ADDR`。
  - 与首次出错写入之后的写入配对的最多两次读取传输可被抑制，且可增加 `READ_ADDR`

上述段落中的“配对”是指写入访问，即写入来自特定读取传输的数据，或反之亦然。DMA 始终按配对方式调度读写访问。

停机行为的细微差异源于在途传输的缓冲以及读写总线端口的并行运行。发生总线错误后，`READ_ADDR`/`WRITE_ADDR` 的值可能略超首次出错的地址，但此差异有界，通常仍足以诊断故障原因。此外，`READ_ADDR` 和 `WRITE_ADDR` 保证以相同增量超前，因为读写操作始终成对调度。

除上述增量外，`READ_ADDR`/`WRITE_ADDR` 始终指向下一个待写入地址，因此若启用地址递增，总会略超故障地址。

## 12.6.8. 附加功能

### 12.6.8.1. 节奏定时器

该定时器允许约每  $n$  个 `clk_sys` 时钟周期传输一次数据，无需使用外部外围设备 DREQ 触发传输。采用分数 (X/Y) 分频器，并在每个 `clk_sys` 周期内最多产生 1 次请求。

RP2350 中提供 4 个定时器。每个 DMA 通道可在 `CTRL.TREQ_SEL` 中选择其中任意一个定时器。有一个寄存器用于配置每个定时器 (TIMER0 至 TIMER3) 的步进系数。

每个定时器的安全级别由 `SECCFG_MISC` 中的寄存器字段定义。该字段规定了配置该定时器所需的最低总线安全级别（较低级别将导致总线故障），以及观察该定时器 TREQ 所需的最低通道安全级别。

### 12.6.8.2. CRC 计算

DMA 可监视经数据 FIFO 的指定通道数据，并基于该数据计算校验和。该过程纯属被动：硬件不会更改数据，仅进行监控。

该功能通过 `SNIFF_CTRL` 和 `SNIFF_DATA` 寄存器控制，并可通过 `CTRL.SNIFF_EN` 字段针对每次 DMA 传输单独启用或禁用。

由于该硬件无法对 FIFO 施加背压，必须跟上 DMA 的最大传输速率，即每个时钟 32 位。

支持的校验和如下：

- CRC-32，MSB 优先和 LSB 优先
- CRC-16-CCITT，MSB 优先和 LSB 优先
- 简单求和（加至 32 位累加器）
- 偶校验

结果寄存器为可读可写，允许设置初始种子值。

结果寄存器支持位/字节操作，以协助特定使用场景：

- 位反转
- 位序反转
- 字节交换

这些操作不会影响 CRC 计算，仅会改变结果寄存器中数据的呈现形式。

嗅探器的安全级别由 `SECCFG_MISC.SNIFF_S` 和 `SECCFG_MISC.SNIFF_P` 位配置。此配置决定访问嗅探器控制寄存器时所需的小总线安全级别，以及嗅探器可监视的最大通道安全级别。

### 12.6.8.3 通道中止

通道可能进入不可恢复的状态。如果指令要求传输的数据量超过外围设备的请求，通道将无法完成。清除 `CTRL.EN` 位会暂停通道，但不能解决该问题。正常情况下不会发生此问题，但必须具备在不对整个 DMA 模块进行硬复位的情况下恢复的机制。

在此情况下，应使用 `CHAN_ABORT` 寄存器强制通道提前完成。每个通道对应一个位。向对应位写入 1 将终止该通道。此操作清除传输计数器并使通道进入非活动状态。

触发中止时，通道可能正处于读写管理器之间的总线传输过程中。这些传输无法撤销。直到这些传输完成且通道达到安全状态，**CTRL.BUSY**标志位保持高电平。这通常只需几个周期。通道在其**CTRL.BUSY**标志取消置位前不得重新启动。当旧序列的传输仍在进行中时启动新的传输序列，将导致不可预测的行为。

中止一个或多个处于未知状态的通道的操作序列（同时兼顾RP2350-E5中描述的行为）如下：

1. 清除所有需中止通道的**EN**位，并禁用**CHAIN\_TO**。
2. 向**CHAN\_ABORT**寄存器写入对应通道的位图。
3. 轮询**ABORT**寄存器，直至先前写入的所有位清除。

中止涉及**CHAIN\_TO**的通道时，建议同时中止该链中的所有其他通道。

#### 12.6.8.4. 调试

每个DMA通道均配有调试寄存器，用于显示dreq计数器**DBG\_CTDREQ**及下一个传输计数**DBG\_TCR**。如有需要，这些寄存器亦可用于重置DMA通道。

### 12.6.9. 示例用例

#### 12.6.9.1. 使用中断重新配置通道

当通道完成一段传输后，即变得可用于执行更多传输。软件检测到通道不再忙碌，随即重新配置并重新启动该通道。一种方法是轮询**CTRL\_BUSY**位，直到通道完成，但这会丧失DMA的一个关键优势，即无需与处理器同步操作。通过设置INTE0至INTE3中的相应位，可以指示DMA在特定通道完成时触发其四个中断请求线之一。您无需反复查询通道状态，而是会主动接收到通知。

##### ① 注意

四个系统中断线允许将不同通道的完成中断路由到不同核心，或在同一核心上优先中断更为关键的通道，且支持针对不同安全域的通道中断。

当中断触发时，处理器可配置为中断当前任务，调用用户指定的处理程序。该处理程序可重新配置并重启通道。当处理程序退出时，处理器将恢复运行中断前台代码。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/dma/channel\\_irq/channel\\_irq.c](https://github.com/raspberrypi/pico-examples/blob/master/dma/channel_irq/channel_irq.c) 第35至52行

```

35 void dma_handler() {
36 static int pwm_level = 0;
37 static uint32_t wavetable[N_PWM_LEVELS];
38 static bool first_run = true;
39 // 条目编号`i`具有`i`个一位和`~(32 - i)`个零位。
40 if (first_run) {
41 first_run = false;
42 for (int i = 0; i < N_PWM_LEVELS; ++i)
43 wavetable[i] = ~(~0u << i);
44 }
45 }
```

```

46 // 清除中断请求。
47 dma_hw->ints0 = 1u << dma_chan;
48 // 赋予通道新的波形表条目进行读取，并重新触发其运行
49 dma_channel_set_read_addr(dma_chan, &wavetable[pwm_level], true);
50
51 pwm_level = (pwm_level + 1) % N_PWM_LEVELS;
52 }

```

在多数情况下，大部分配置可于通道首次启动时完成。如此，中断处理程序中仅需重新配置地址和传输长度。

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/dma/channel\\_irq/channel\\_irq.c](https://github.com/raspberrypi/pico-examples/blob/master/dma/channel_irq/channel_irq.c) 第54行至94行

```

54 int main() {
55 #ifndef PICO_DEFAULT_LED_PIN
56 //warning dma/channel_irq 示例需配备常规LED的开发板
57 #else
58 // 配置PIO状态机以实现位串行化
59 uint offset = pio_add_program(pio0, &pio_serialiser_program);
60 pio_serialiser_program_init(pio0, 0, offset, PICO_DEFAULT_LED_PIN, PIO_SERIAL_CLKDIV);
61
62 // 配置一个通道以重复写入相同数据（32位）到 PIO0
63 // SM0 的 TX FIFO，由该外设的数据请求信号节奏控制。
64 dma_chan = dma_claim_unused_channel(true);
65 dma_channel_config c = dma_channel_get_default_config(dma_chan);
66 channel_config_set_transfer_data_size(&c, DMA_SIZE_32);
67 channel_config_set_read_increment(&c, false);
68 channel_config_set_dreq(&c, DREQ_PI00_TX0);
69
70 dma_channel_configure(
71 dma_chan,
72 &c,
73 &pio0_hw->txf[0], // 写地址（仅需设置一次）
74 NULL, // 暂不提供读取地址
75 PWM_REPEAT_COUNT, // 多次写入相同数值，随后停止并触发中断
76 false // 暂不启动
77);
78
79 // 指示 DMA 在通道完成一个块传输时激活 IRQ 线 0
80 dma_channel_set_irq0_enabled(dma_chan, true);
81
82 // 配置处理器在 DMA IRQ 0 断言时执行dma_handler()
83 irq_set_exclusive_handler(DMA_IRQ_0, dma_handler);
84 irq_set_enabled(DMA_IRQ_0, true);
85
86 // 手动调用一次处理程序以触发第一次传输
87 dma_handler();
88
89 // 从此开始，其他所有操作均由中断驱动。处理器
90 // 有时间考虑提前退休——也许开家面包店？
91 while (true)
92 tight_loop_contents();
93 #endif
94 }

```

该技术的一个缺点是，直到通道完成最后一次传输后，才开始重新配置通道。若处理器中断活动频繁，这段时间可能较长，传输之间存在较大间隙。这使得持续保持高数据吞吐率变得困难。

通过使用两个通道及其 CHAIN\_TO 字段交叉设置实现此目的，使通道A在完成时触发通道B，反之亦然。在任何时间点，始终有一个通道正在传输数据。另一个通道要么已经

被配置为在当前传输结束后立即启动下一次传输，或处于重新配置过程中。通道A完成后立即启动通道B上排队的传输，同时触发中断，中断处理程序重新配置通道A，以确保其在通道B完成后准备就绪。

### 12.6.9.2. DMA控制块

经常需要将多个较小缓冲区汇总后发送至同一外设。为满足此需求，RP2350 DMA可执行无需处理器干预的长而复杂的传输序列。一个通道持续重新配置第二通道，第二通道每完成一块传输即重新启动第一个通道。

由于第一个DMA通道直接将数据从内存传输到第二通道的控制寄存器，内存中的控制块格式必须与这些寄存器相匹配。每次写入的最后一个寄存器均为触发寄存器之一（第12.6.3.1节），该寄存器将启动第二通道执行其编程的传输块。寄存器别名（第12.6.3.1节）为块布局提供一定灵活性，更重要的是允许省略某些寄存器，从而减少内存占用并加快加载速度。

本示例展示了如何通过重新编程TRANS\_COUNT和READ\_ADDR\_TRIG收集多个缓冲区并传输至UART：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/dma/control\\_blocks/control\\_blocks.c](https://github.com/raspberrypi/pico-examples/blob/master/dma/control_blocks/control_blocks.c)

```

1 /**
2 * 版权声明 (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4 * SPDX许可证标识符: BSD-3-Clause
5 */
6
7 // 使用两个DMA通道执行一系列编程的数据传输至
8 // UART (一种数据收集操作)。一个通道负责传输
9 // 实际数据，另一个通道则不断重新编程该通道。
10
11 #include <stdio.h>
12 #include "pico/stl.h"
13 #include "hardware/dma.h"
14 #include "hardware/structs/uart.h"
15
16 // 这些缓冲区将依次通过 DMA 传输到 UART。
17
18 const char word0[] = "Transferring ";
19 const char word1[] = "one ";
20 const char word2[] = "word ";
21 const char word3[] = "at ";
22 const char word4[] = "a ";
23 const char word5[] = "time.\n";
24
25 // 注意字段顺序：长度必须置于
26 // 读取地址之前，因为控制通道将写入数据通道别名3中的最后
27 // 两个寄存器：
28 // +0x0 +0x4 +0x8 +0xC (触发)
29 // 别名 0: READ_ADDR WRITE_ADDR TRANS_COUNT CTRL
30 // 别名 1: CTRL READ_ADDR WRITE_ADDR TRANS_COUNT
31 // 别名 2: CTRL TRANS_COUNT READ_ADDR WRITE_ADDR
32 // 别名 3: CTRL WRITE_ADDR TRANS_COUNT READ_ADDR
33 //
34 // 这将设定数据通道的传输计数和读取地址，
35 // 并触发该通道。一旦数据通道执行完成，它将重启
36 // 控制通道 (通过 CHAIN_TO) 以将下一组两个字加载至其控制寄存器。
37 // registers.
38
39 const struct {uint32_t len; const char *data;} control_blocks[] = {

```

```

40 {count_of(word0) - 1, word0}, // 跳过空终止符
41 {count_of(word1) - 1, word1},
42 {count_of(word2) - 1, word2},
43 {count_of(word3) - 1, word3},
44 {count_of(word4) - 1, word4},
45 {count_of(word5) - 1, word5},
46 {0, NULL} // 空触发器用于结束链。
47 };
48
49 int main() {
50 #ifndef uart_default
51 #warning dma/control_blocks 示例需要 UART
52 #else
53 stdio_init_all();
54 puts("DMA 控制块示例: ");
55
56 // ctrl_chan 将控制块加载到 data_chan，后者执行这些控制块。
57 int ctrl_chan = dma_claim_unused_channel(true);
58 int data_chan = dma_claim_unused_channel(true);
59
60 // 控制通道将两个字传输至数据通道的控制寄存器，随后停止。
61 // registers, then halts. 写地址沿两个字
62 // (八字节) 边界循环，因此控制通道写入相同的两个字
63 // 下次触发时的寄存器数量。
64
65 dma_channel_config c = dma_channel_get_default_config(ctrl_chan);
66 channel_config_set_transfer_data_size(&c, DMA_SIZE_32);
67 channel_config_set_read_increment(&c, true);
68 channel_config_set_write_increment(&c, true);
69 channel_config_set_ring(&c, true, 3); // 写指针采用 1 << 3 字节边界
70
71 dma_channel_configure(
72 ctrl_chan,
73 &c,
74 &dma_hw->ch[data_chan].a13_transfer_count, // 初始写地址
75 &control_blocks[0], // 初始读地址
76 2, // 每个控制块执行后暂停
77 false // 暂不启动
78);
79
80 // 数据通道设置为写入 UART FIFO (由UART的TX数据请求信号驱动) ,
81 // 完成后链至控制通道
82 // 。 控制通道设置新的读地址及
83 // 数据长度，并重新触发数据通道。
84
85 c = dma_channel_get_default_config(data_chan);
86 channel_config_set_transfer_data_size(&c, DMA_SIZE_8);
87 channel_config_set_dreq(&c, uart_get_dreq(uart_default, true));
88 // 当 data_chan 完成时触发 ctrl_chan
89 channel_config_set_chain_to(&c, ctrl_chan);
90 // 当向触发寄存器写入 0 (链结束) 时，触发 IRQ 标志：
91 channel_config_set_irq_quiet(&c, true);
92
93 dma_channel_configure(
94 data_chan,
95 &c,
96 &uart_get_hw(uart_default)->dr,
97 NULL, // 初始读取地址及传输计数无关紧要；
98 0, // 控制通道将每次重新编程它们。
99 false // 暂时不启动。
100);
101
102 // 所有准备就绪。通知控制通道加载第一个
103 // 控制块。从此开始一切自动完成。

```

```

104 dma_start_channel_mask(1u << ctrl_chan);
105
106 // 数据通道在收到空触发时将置位其 IRQ 标志,
107 // 表示控制块列表已结束。我们将仅等待108// IRQ标志，而不设置中断处理程序。
108
109 while (!(dma_hw->intr & 1u << data_chan))
110 tight_loop_contents();
111 dma_hw->ints0 = 1u << data_chan;
112
113 puts("DMA完成。");
114 #endif
115 }

```

## 12.6.10. 寄存器列表

DMA 寄存器起始地址为 **0x50000000**(在SDK中定义为DMA\_BASE)。

表1147.  
DMA寄存器列表

偏移量	名称	说明
0x000	CH0_READ_ADDR	DMA通道0读取地址指针
0x004	CH0_WRITE_ADDR	DMA通道0写入地址指针
0x008	CH0_TRANS_COUNT	DMA通道0传输计数
0x00c	CH0_CTRL_TRIG	DMA通道0控制与状态
0x010	CH0_AL1_CTRL	通道0 CTRL寄存器别名
0x014	CH0_AL1_READ_ADDR	通道0 READ_ADDR寄存器别名
0x018	CH0_AL1_WRITE_ADDR	通道0 WRITE_ADDR寄存器别名
0x01c	CH0_AL1_TRANS_COUNT_TRIG	通道0 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x020	CH0_AL2_CTRL	通道0 CTRL寄存器别名
0x024	CH0_AL2_TRANS_COUNT	通道0 TRANS_COUNT寄存器别名
0x028	CH0_AL2_READ_ADDR	通道0 READ_ADDR寄存器别名
0x02c	CH0_AL2_WRITE_ADDR_TRIG	通道0 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x030	CH0_AL3_CTRL	通道0 CTRL寄存器别名
0x034	CH0_AL3_WRITE_ADDR	通道0 WRITE_ADDR寄存器别名
0x038	CH0_AL3_TRANS_COUNT	通道0 TRANS_COUNT寄存器别名
0x03c	CH0_AL3_READ_ADDR_TRIG	通道0 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x040	CH1_READ_ADDR	DMA 通道 1 读地址指针
0x044	CH1_WRITE_ADDR	DMA 通道 1 写地址指针
0x048	CH1_TRANS_COUNT	DMA 通道 1 传输计数
0x04c	CH1_CTRL_TRIG	DMA 通道 1 控制与状态

偏移量	名称	说明
0x050	CH1_AL1_CTRL	通道 1 CTRL 寄存器别名
0x054	CH1_AL1_READ_ADDR	通道 1 READ_ADDR 寄存器别名
0x058	CH1_AL1_WRITE_ADDR	通道 1 WRITE_ADDR 寄存器别名
0x05c	CH1_AL1_TRANS_COUNT_TRIG	通道 1 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x060	CH1_AL2_CTRL	通道 1 CTRL 寄存器别名
0x064	CH1_AL2_TRANS_COUNT	通道 1 TRANS_COUNT 寄存器别名
0x068	CH1_AL2_READ_ADDR	通道 1 READ_ADDR 寄存器别名
0x06c	CH1_AL2_WRITE_ADDR_TRIG	通道 1 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x070	CH1_AL3_CTRL	通道 1 CTRL 寄存器别名
0x074	CH1_AL3_WRITE_ADDR	通道 1 WRITE_ADDR 寄存器别名
0x078	CH1_AL3_TRANS_COUNT	通道 1 TRANS_COUNT 寄存器别名
0x07c	CH1_AL3_READ_ADDR_TRIG	通道 1 READ_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x080	CH2_READ_ADDR	DMA 通道 2 读地址指针
0x084	CH2_WRITE_ADDR	DMA 通道 2 写地址指针
0x088	CH2_TRANS_COUNT	DMA 通道 2 传输计数
0x08c	CH2_CTRL_TRIG	DMA 通道 2 控制与状态
0x090	CH2_AL1_CTRL	通道 2 CTRL 寄存器别名
0x094	CH2_AL1_READ_ADDR	通道 2 READ_ADDR 寄存器别名
0x098	CH2_AL1_WRITE_ADDR	通道 2 WRITE_ADDR 寄存器别名
0x09c	CH2_AL1_TRANS_COUNT_TRIG	通道 2 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x0a0	CH2_AL2_CTRL	通道 2 CTRL 寄存器别名
0x0a4	CH2_AL2_TRANS_COUNT	通道 2 TRANS_COUNT 寄存器别名
0x0a8	CH2_AL2_READ_ADDR	通道 2 READ_ADDR 寄存器别名
0x0ac	CH2_AL2_WRITE_ADDR_TRIG	通道 2 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x0b0	CH2_AL3_CTRL	通道 2 CTRL 寄存器别名
0x0b4	CH2_AL3_WRITE_ADDR	通道 2 WRITE_ADDR 寄存器别名
0x0b8	CH2_AL3_TRANS_COUNT	通道 2 TRANS_COUNT 寄存器别名
0x0bc	CH2_AL3_READ_ADDR_TRIG	通道 2 READ_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。

偏移量	名称	说明
0x0c0	CH3_READ_ADDR	DMA 通道 3 读地址指针
0x0c4	CH3_WRITE_ADDR	DMA 通道 3 写地址指针
0x0c8	CH3_TRANS_COUNT	DMA 通道 3 传输计数
0x0cc	CH3_CTRL_TRIG	DMA 通道 3 控制与状态
0x0d0	CH3_AL1_CTRL	通道 3 CTRL 寄存器别名
0x0d4	CH3_AL1_READ_ADDR	通道 3 READ_ADDR 寄存器别名
0x0d8	CH3_AL1_WRITE_ADDR	通道 3 WRITE_ADDR 寄存器别名
0x0dc	CH3_AL1_TRANS_COUNT_TRIG	通道 3 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x0e0	CH3_AL2_CTRL	通道 3 CTRL 寄存器别名
0x0e4	CH3_AL2_TRANS_COUNT	通道 3 TRANS_COUNT 寄存器别名
0x0e8	CH3_AL2_READ_ADDR	通道 3 READ_ADDR 寄存器别名
0x0ec	CH3_AL2_WRITE_ADDR_TRIG	通道 3 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x0f0	CH3_AL3_CTRL	通道 3 CTRL 寄存器别名
0x0f4	CH3_AL3_WRITE_ADDR	通道 3 WRITE_ADDR 寄存器别名
0x0f8	CH3_AL3_TRANS_COUNT	通道 3 TRANS_COUNT 寄存器别名
0x0fc	CH3_AL3_READ_ADDR_TRIG	通道 3 READ_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x100	CH4_READ_ADDR	DMA 通道 4 读取地址指针
0x104	CH4_WRITE_ADDR	DMA 通道 4 写入地址指针
0x108	CH4_TRANS_COUNT	DMA 通道 4 传输计数
0x10c	CH4_CTRL_TRIG	DMA 通道 4 控制与状态
0x110	CH4_AL1_CTRL	通道 4 CTRL 寄存器别名
0x114	CH4_AL1_READ_ADDR	通道 4 READ_ADDR 寄存器别名
0x118	CH4_AL1_WRITE_ADDR	通道 4 WRITE_ADDR 寄存器别名
0x11c	CH4_AL1_TRANS_COUNT_TRIG	通道 4 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x120	CH4_AL2_CTRL	通道 4 CTRL 寄存器别名
0x124	CH4_AL2_TRANS_COUNT	通道 4 TRANS_COUNT 寄存器别名
0x128	CH4_AL2_READ_ADDR	通道 4 READ_ADDR 寄存器别名
0x12c	CH4_AL2_WRITE_ADDR_TRIG	通道 4 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x130	CH4_AL3_CTRL	通道 4 CTRL 寄存器别名

偏移量	名称	说明
0x134	CH4_AL3_WRITE_ADDR	通道4 WRITE_ADDR寄存器别名
0x138	CH4_AL3_TRANS_COUNT	通道4 TRANS_COUNT寄存器别名
0x13c	CH4_AL3_READ_ADDR_TRIG	通道4 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x140	CH5_READ_ADDR	DMA通道5读取地址指针
0x144	CH5_WRITE_ADDR	DMA通道5写入地址指针
0x148	CH5_TRANS_COUNT	DMA通道5传输计数
0x14c	CH5_CTRL_TRIG	DMA通道5控制与状态
0x150	CH5_AL1_CTRL	通道5 CTRL寄存器的别名
0x154	CH5_AL1_READ_ADDR	通道5 READ_ADDR寄存器的别名
0x158	CH5_AL1_WRITE_ADDR	通道5 WRITE_ADDR寄存器的别名
0x15c	CH5_AL1_TRANS_COUNT_TRIG	通道5 TRANS_COUNT寄存器的别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x160	CH5_AL2_CTRL	通道5 CTRL寄存器的别名
0x164	CH5_AL2_TRANS_COUNT	通道5 TRANS_COUNT寄存器的别名
0x168	CH5_AL2_READ_ADDR	通道5 READ_ADDR寄存器的别名
0x16c	CH5_AL2_WRITE_ADDR_TRIG	通道5 WRITE_ADDR寄存器的别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x170	CH5_AL3_CTRL	通道5 CTRL寄存器的别名
0x174	CH5_AL3_WRITE_ADDR	通道5 WRITE_ADDR寄存器的别名
0x178	CH5_AL3_TRANS_COUNT	通道5 TRANS_COUNT寄存器的别名
0x17c	CH5_AL3_READ_ADDR_TRIG	通道5 READ_ADDR寄存器的别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x180	CH6_READ_ADDR	DMA通道6读取地址指针
0x184	CH6_WRITE_ADDR	DMA通道6写入地址指针
0x188	CH6_TRANS_COUNT	DMA通道6传输计数
0x18c	CH6_CTRL_TRIG	DMA通道6控制与状态寄存器
0x190	CH6_AL1_CTRL	通道6 CTRL寄存器的别名
0x194	CH6_AL1_READ_ADDR	通道6 READ_ADDR寄存器的别名
0x198	CH6_AL1_WRITE_ADDR	通道6 WRITE_ADDR寄存器的别名
0x19c	CH6_AL1_TRANS_COUNT_TRIG	通道6 TRANS_COUNT寄存器的别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x1a0	CH6_AL2_CTRL	通道6 CTRL寄存器的别名
0x1a4	CH6_AL2_TRANS_COUNT	通道6 TRANS_COUNT寄存器的别名

偏移量	名称	说明
0x1A8	CH6_AL2_READ_ADDR	通道6 READ_ADDR寄存器的别名
0x1AC	CH6_AL2_WRITE_ADDR_TRIG	通道6 WRITE_ADDR寄存器的别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x1b0	CH6_AL3_CTRL	通道6 CTRL寄存器的别名
0x1b4	CH6_AL3_WRITE_ADDR	通道6 WRITE_ADDR寄存器的别名
0x1b8	CH6_AL3_TRANS_COUNT	通道6 TRANS_COUNT寄存器的别名
0x1bc	CH6_AL3_READ_ADDR_TRIG	通道6 READ_ADDR寄存器的别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x1c0	CH7_READ_ADDR	DMA通道7读取地址指针
0x1c4	CH7_WRITE_ADDR	DMA通道7写地址指针
0x1c8	CH7_TRANS_COUNT	DMA通道7传输计数
0x1cc	CH7_CTRL_TRIG	DMA通道7控制与状态
0x1d0	CH7_AL1_CTRL	通道7 CTRL寄存器别名
0x1d4	CH7_AL1_READ_ADDR	通道7 READ_ADDR寄存器别名
0x1d8	CH7_AL1_WRITE_ADDR	通道7 WRITE_ADDR寄存器别名
0x1dc	CH7_AL1_TRANS_COUNT_TRIG	通道7 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x1e0	CH7_AL2_CTRL	通道7 CTRL寄存器别名
0x1e4	CH7_AL2_TRANS_COUNT	通道7 TRANS_COUNT寄存器别名
0x1e8	CH7_AL2_READ_ADDR	通道7 READ_ADDR寄存器别名
0x1ec	CH7_AL2_WRITE_ADDR_TRIG	通道7 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x1f0	CH7_AL3_CTRL	通道7 CTRL寄存器别名
0x1f4	CH7_AL3_WRITE_ADDR	通道7 WRITE_ADDR寄存器别名
0x1f8	CH7_AL3_TRANS_COUNT	通道7 TRANS_COUNT寄存器别名
0x1fc	CH7_AL3_READ_ADDR_TRIG	通道7 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x200	CH8_READ_ADDR	DMA通道8读地址指针
0x204	CH8_WRITE_ADDR	DMA通道8写地址指针
0x208	CH8_TRANS_COUNT	DMA通道8传输计数
0x20c	CH8_CTRL_TRIG	DMA通道8控制与状态
0x210	CH8_AL1_CTRL	通道8 CTRL寄存器别名
0x214	CH8_AL1_READ_ADDR	通道8 READ_ADDR寄存器别名
0x218	CH8_AL1_WRITE_ADDR	通道8 WRITE_ADDR寄存器别名

偏移量	名称	说明
0x21c	CH8_AL1_TRANS_COUNT_TRIG	通道8 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x220	CH8_AL2_CTRL	通道8 CTRL寄存器别名
0x224	CH8_AL2_TRANS_COUNT	通道8 TRANS_COUNT寄存器别名
0x228	CH8_AL2_READ_ADDR	通道8 READ_ADDR寄存器别名
0x22c	CH8_AL2_WRITE_ADDR_TRIG	通道8 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x230	CH8_AL3_CTRL	通道8 CTRL寄存器别名
0x234	CH8_AL3_WRITE_ADDR	通道8 WRITE_ADDR寄存器别名
0x238	CH8_AL3_TRANS_COUNT	通道8 TRANS_COUNT寄存器别名
0x23c	CH8_AL3_READ_ADDR_TRIG	通道8 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x240	CH9_READ_ADDR	DMA通道9读取地址指针
0x244	CH9_WRITE_ADDR	DMA通道9写入地址指针
0x248	CH9_TRANS_COUNT	DMA通道9传输计数
0x24c	CH9_CTRL_TRIG	DMA通道9控制与状态寄存器
0x250	CH9_AL1_CTRL	通道9 CTRL寄存器别名
0x254	CH9_AL1_READ_ADDR	通道9 READ_ADDR寄存器别名
0x258	CH9_AL1_WRITE_ADDR	通道9 WRITE_ADDR寄存器别名
0x25c	CH9_AL1_TRANS_COUNT_TRIG	通道9 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x260	CH9_AL2_CTRL	通道9 CTRL寄存器别名
0x264	CH9_AL2_TRANS_COUNT	通道9 TRANS_COUNT寄存器别名
0x268	CH9_AL2_READ_ADDR	通道9 READ_ADDR寄存器别名
0x26c	CH9_AL2_WRITE_ADDR_TRIG	通道9 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x270	CH9_AL3_CTRL	通道9 CTRL寄存器别名
0x274	CH9_AL3_WRITE_ADDR	通道9 WRITE_ADDR寄存器别名
0x278	CH9_AL3_TRANS_COUNT	通道9 TRANS_COUNT寄存器别名
0x27c	CH9_AL3_READ_ADDR_TRIG	通道9 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x280	CH10_READ_ADDR	DMA通道10读取地址指针
0x284	CH10_WRITE_ADDR	DMA通道10写入地址指针
0x288	CH10_TRANS_COUNT	DMA通道10传输计数

偏移量	名称	说明
0x28c	CH10_CTRL_TRIG	DMA通道10控制与状态寄存器
0x290	CH10_AL1_CTRL	通道10 CTRL寄存器别名
0x294	CH10_AL1_READ_ADDR	通道10 READ_ADDR寄存器别名
0x298	CH10_AL1_WRITE_ADDR	通道10 WRITE_ADDR寄存器别名
0x29c	CH10_AL1_TRANS_COUNT_TRIG	通道10 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x2a0	CH10_AL2_CTRL	通道10 CTRL寄存器别名
0x2a4	CH10_AL2_TRANS_COUNT	通道10 TRANS_COUNT寄存器别名
0x2a8	CH10_AL2_READ_ADDR	通道10 READ_ADDR寄存器别名
0x2ac	CH10_AL2_WRITE_ADDR_TRIG	通道10 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x2b0	CH10_AL3_CTRL	通道10 CTRL寄存器别名
0x2b4	CH10_AL3_WRITE_ADDR	通道10 WRITE_ADDR寄存器别名
0x2b8	CH10_AL3_TRANS_COUNT	通道10 TRANS_COUNT寄存器别名
0x2bc	CH10_AL3_READ_ADDR_TRIG	通道10 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x2c0	CH11_READ_ADDR	DMA通道11读取地址指针
0x2c4	CH11_WRITE_ADDR	DMA通道11写入地址指针
0x2c8	CH11_TRANS_COUNT	DMA通道11传输计数
0x2cc	CH11_CTRL_TRIG	DMA通道11控制与状态
0x2d0	CH11_AL1_CTRL	通道11 CTRL寄存器别名
0x2d4	CH11_AL1_READ_ADDR	通道11 READ_ADDR寄存器别名
0x2d8	CH11_AL1_WRITE_ADDR	通道11 WRITE_ADDR寄存器别名
0x2dc	CH11_AL1_TRANS_COUNT_TRIG	通道11 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x2e0	CH11_AL2_CTRL	通道11 CTRL寄存器别名
0x2e4	CH11_AL2_TRANS_COUNT	通道11 TRANS_COUNT寄存器别名
0x2e8	CH11_AL2_READ_ADDR	通道11 READ_ADDR寄存器别名
0x2ec	CH11_AL2_WRITE_ADDR_TRIG	通道11 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x2f0	CH11_AL3_CTRL	通道11 CTRL寄存器别名
0x2f4	CH11_AL3_WRITE_ADDR	通道11 WRITE_ADDR寄存器别名
0x2f8	CH11_AL3_TRANS_COUNT	通道11 TRANS_COUNT寄存器别名

偏移量	名称	说明
0x2fc	CH11_AL3_READ_ADDR_TRIG	通道11 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x300	CH12_READ_ADDR	DMA通道12读取地址指针
0x304	CH12_WRITE_ADDR	DMA通道12写入地址指针
0x308	CH12_TRANS_COUNT	DMA通道12传输计数
0x30c	CH12_CTRL_TRIG	DMA通道12控制与状态
0x310	CH12_AL1_CTRL	通道12 CTRL寄存器别名
0x314	CH12_AL1_READ_ADDR	通道12 READ_ADDR寄存器别名
0x318	CH12_AL1_WRITE_ADDR	通道12 WRITE_ADDR寄存器别名
0x31c	CH12_AL1_TRANS_COUNT_TRIG	通道12 TRANS_COUNT寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x320	CH12_AL2_CTRL	通道12 CTRL寄存器别名
0x324	CH12_AL2_TRANS_COUNT	通道12 TRANS_COUNT寄存器别名
0x328	CH12_AL2_READ_ADDR	通道12 READ_ADDR寄存器别名
0x32c	CH12_AL2_WRITE_ADDR_TRIG	通道12 WRITE_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x330	CH12_AL3_CTRL	通道12 CTRL寄存器别名
0x334	CH12_AL3_WRITE_ADDR	通道12 WRITE_ADDR寄存器别名
0x338	CH12_AL3_TRANS_COUNT	通道12 TRANS_COUNT寄存器别名
0x33c	CH12_AL3_READ_ADDR_TRIG	通道12 READ_ADDR寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x340	CH13_READ_ADDR	DMA通道13读取地址指针
0x344	CH13_WRITE_ADDR	DMA 通道 13 写地址指针
0x348	CH13_TRANS_COUNT	DMA 通道 13 传输计数
0x34c	CH13_CTRL_TRIG	DMA 通道 13 控制与状态
0x350	CH13_AL1_CTRL	通道 13 CTRL 寄存器别名
0x354	CH13_AL1_READ_ADDR	通道 13 READ_ADDR 寄存器别名
0x358	CH13_AL1_WRITE_ADDR	通道 13 WRITE_ADDR 寄存器别名
0x35c	CH13_AL1_TRANS_COUNT_TRIG	通道 13 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x360	CH13_AL2_CTRL	通道 13 CTRL 寄存器别名
0x364	CH13_AL2_TRANS_COUNT	通道 13 TRANS_COUNT 寄存器别名
0x368	CH13_AL2_READ_ADDR	通道 13 READ_ADDR 寄存器别名

偏移量	名称	说明
0x36c	CH13_AL2_WRITE_ADDR_TRIG	通道 13 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x370	CH13_AL3_CTRL	通道 13 CTRL 寄存器别名
0x374	CH13_AL3_WRITE_ADDR	通道 13 WRITE_ADDR 寄存器别名
0x378	CH13_AL3_TRANS_COUNT	通道 13 TRANS_COUNT 寄存器别名
0x37c	CH13_AL3_READ_ADDR_TRIG	通道 13 READ_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x380	CH14_READ_ADDR	DMA 通道 14 读取地址指针
0x384	CH14_WRITE_ADDR	DMA 通道 14 写入地址指针
0x388	CH14_TRANS_COUNT	DMA 通道 14 传输计数
0x38c	CH14_CTRL_TRIG	DMA 通道 14 控制与状态
0x390	CH14_AL1_CTRL	通道 14 CTRL 寄存器别名
0x394	CH14_AL1_READ_ADDR	通道 14 READ_ADDR 寄存器别名
0x398	CH14_AL1_WRITE_ADDR	通道 14 WRITE_ADDR 寄存器别名
0x39c	CH14_AL1_TRANS_COUNT_TRIG	通道 14 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x3a0	CH14_AL2_CTRL	通道 14 CTRL 寄存器别名
0x3a4	CH14_AL2_TRANS_COUNT	通道 14 TRANS_COUNT 寄存器别名
0x3a8	CH14_AL2_READ_ADDR	通道 14 READ_ADDR 寄存器别名
0x3ac	CH14_AL2_WRITE_ADDR_TRIG	通道 14 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x3b0	CH14_AL3_CTRL	通道 14 CTRL 寄存器别名
0x3b4	CH14_AL3_WRITE_ADDR	通道 14 WRITE_ADDR 寄存器别名
0x3b8	CH14_AL3_TRANS_COUNT	通道 14 TRANS_COUNT 寄存器别名
0x3bc	CH14_AL3_READ_ADDR_TRIG	通道 14 READ_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。
0x3c0	CH15_READ_ADDR	DMA 通道 15 读地址指针
0x3c4	CH15_WRITE_ADDR	DMA 通道 15 写地址指针
0x3c8	CH15_TRANS_COUNT	DMA 通道 15 传输计数
0x3cc	CH15_CTRL_TRIG	DMA 通道 15 控制与状态
0x3d0	CH15_AL1_CTRL	通道 15 CTRL 寄存器别名
0x3d4	CH15_AL1_READ_ADDR	通道 15 READ_ADDR 寄存器别名
0x3d8	CH15_AL1_WRITE_ADDR	通道 15 WRITE_ADDR 寄存器别名

偏移量	名称	说明
0x3dc	CH15_AL1_TRANS_COUNT_TRIG	通道15 TRANS_COUNT 寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x3e0	CH15_AL2_CTRL	通道15 CTRL 寄存器别名
0x3e4	CH15_AL2_TRANS_COUNT	通道15 TRANS_COUNT 寄存器别名
0x3e8	CH15_AL2_READ_ADDR	通道15 READ_ADDR 寄存器别名
0x3ec	CH15_AL2_WRITE_ADDR_TRIG	通道15 WRITE_ADDR 寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x3f0	CH15_AL3_CTRL	通道15 CTRL 寄存器别名
0x3f4	CH15_AL3_WRITE_ADDR	通道15 WRITE_ADDR 寄存器别名
0x3f8	CH15_AL3_TRANS_COUNT	通道15 TRANS_COUNT 寄存器别名
0x3fc	CH15_AL3_READ_ADDR_TRIG	通道15 READ_ADDR 寄存器别名 该寄存器为触发寄存器（0xc）。写入非零值将重载通道计数器并启动该通道。
0x400	中断	中断状态（原始）
0x404	INTE0	IRQ0 中断使能
0x408	INTF0	强制中断
0x40c	INTS0	IRQ 0 的中断状态
0x414	INTE1	IRQ 1 的中断使能
0x418	INTF1	强制中断
0x41c	INTS1	IRQ 1 的中断状态
0x424	INTE2	IRQ 2 的中断使能
0x428	INTF2	强制中断
0x42c	INTS2	IRQ 2 的中断状态
0x434	INTE3	IRQ 3 的中断使能
0x438	INTF3	强制中断
0x43c	INTS3	IRQ 3 的中断状态
0x440	TIMER0	节奏计时器（生成周期性 TREQ）
0x444	TIMER1	节奏计时器（生成周期性 TREQ）
0x448	TIMER2	节奏计时器（生成周期性 TREQ）
0x44c	TIMER3	节奏计时器（生成周期性 TREQ）
0x450	MULTI_CHAN_TRIGGER	同时触发一个或多个通道
0x454	SNIFF_CTRL	嗅探器控制
0x458	SNIFF_DATA	用于嗅探硬件的数据累加器
0x460	FIFO_LEVELS	调试 RAF、WAF、TDF 级别
0x464	CHAN_ABORT	中止一个或多个通道上正在进行的传输序列

偏移量	名称	说明
0x468	N_CHANNELS	该 DMA 实例配备的通道数量。 此 DMA 支持最多 16 个硬件通道，但可配置为仅使用一个，以最大限度减少硅片面积。
0x480	SECCFG_CH0	通道 0 的安全级别配置。
0x484	SECCFG_CH1	通道 1 的安全级别配置。
0x488	SECCFG_CH2	通道 2 的安全级别配置。
0x48c	SECCFG_CH3	通道3的安全级别配置。
0x490	SECCFG_CH4	通道4的安全级别配置。
0x494	SECCFG_CH5	通道5的安全级别配置。
0x498	SECCFG_CH6	通道6的安全级别配置。
0x49c	SECCFG_CH7	通道7的安全级别配置。
0x4a0	SECCFG_CH8	通道8的安全级别配置。
0x4a4	SECCFG_CH9	通道9的安全级别配置。
0x4a8	SECCFG_CH10	通道10的安全级别配置。
0x4ac	SECCFG_CH11	通道11的安全级别配置。
0x4b0	SECCFG_CH12	通道12的安全等级配置。
0x4b4	SECCFG_CH13	通道13的安全等级配置。
0x4b8	SECCFG_CH14	通道14的安全等级配置。
0x4bc	SECCFG_CH15	通道15的安全等级配置。
0x4c0	SECCFG_IRQ0	IRQ 0 的安全配置。控制 IRQ 是否允许被非安全或非特权上下文配置，以及是否允许其监视安全或特权通道的中断标志。
0x4c4	SECCFG_IRQ1	IRQ 1 的安全配置。控制 IRQ 是否允许被非安全或非特权上下文配置，以及是否允许其监视安全或特权通道的中断标志。
0x4c8	SECCFG_IRQ2	IRQ 2 的安全配置。控制 IRQ 是否允许被非安全或非特权上下文配置，以及是否允许其监视安全或特权通道的中断标志。
0x4cc	SECCFG_IRQ3	IRQ 3 的安全配置。控制 IRQ 是否允许被非安全或非特权上下文配置，以及是否允许其监视安全或特权通道的中断标志。
0x4d0	SECCFG_MISC	其它安全配置
0x500	MPU_CTRL	DMA MPU 控制寄存器，仅可在特权模式下访问。
0x504	MPU_BAR0	MPU 区域 0 的基址寄存器。仅可在安全且特权模式下写入。
0x508	MPU_LAR0	MPU 区域 0 的限制地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x50c	MPU_BAR1	MPU 区域 1 的基址寄存器。仅可在安全且特权模式下写入。

偏移量	名称	说明
0x510	MPU_LAR1	MPU 区域 1 的限制地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x514	MPU_BAR2	MPU 区域 2 的基址寄存器。仅可在安全且特权模式下写入。
0x518	MPU_LAR2	MPU 区域 2 的限制地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x51c	MPU_BAR3	MPU 区域 3 的基址寄存器。仅可在安全且特权模式下写入。
0x520	MPU_LAR3	MPU 第 3 区域限界地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x524	MPU_BAR4	MPU 第 4 区域基地址寄存器。仅可在安全且特权模式下写入。
0x528	MPU_LAR4	MPU 第 4 区域限界地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x52c	MPU_BAR5	MPU 第 5 区域基地址寄存器。仅可在安全且特权模式下写入。
0x530	MPU_LAR5	MPU 第 5 区域限界地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x534	MPU_BAR6	MPU 第 6 区域基地址寄存器。仅可在安全且特权模式下写入。
0x538	MPU_LAR6	MPU 第 6 区域限界地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x53c	MPU_BAR7	MPU 第 7 区域基地址寄存器。仅可在安全且特权模式下写入。
0x540	MPU_LAR7	MPU 第 7 区域限界地址寄存器。除 P 位外，仅可在安全且特权模式下写入。
0x800	CH0_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x804	CH0_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0x840	CH1_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x844	CH1_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0x880	CH2_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x884	CH2_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度

偏移量	名称	说明
0x8c0	CH3_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x8c4	CH3_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0x900	CH4_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x904	CH4_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0x940	CH5_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x944	CH5_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0x980	CH6_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x984	CH6_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0x9c0	CH7_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0x9c4	CH7_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xa00	CH8_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xa04	CH8_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xa40	CH9_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xa44	CH9_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xa80	CH10_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。

偏移量	名称	说明
0xa84	CH10_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xac0	CH11_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xac4	CH11_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xb00	CH12_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xb04	CH12_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xb40	CH13_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xb44	CH13_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xb80	CH14_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xb84	CH14_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度
0xbc0	CH15_DBG_CTDREQ	读取：获取通道DREQ计数器（即DMA预计可对外设执行的访问次数，防止溢出或下溢）。写入任意值：清零计数器，并使通道重新发起DREQ握手。
0xbc4	CH15_DBG_TCR	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度

## DMA: CH0\_READ\_ADDR, CH1\_READ\_ADDR, ..., CH14\_READ\_ADDR, CH15\_READ\_ADDR 寄存器

偏移量：0x000, 0x040, ..., 0x380, 0x3c0

### 描述

DMA 通道 N 读取地址指针

表 1148  
CH0\_READ\_ADDR,  
CH1\_READ\_ADDR, ...,  
CH14\_READ\_ADDR,  
CH15\_READ\_ADDR  
寄存器

位	描述	类型	复位
31:0	该寄存器在每次读取完成时自动更新。当前值为该通道下一次将要读取的地址。	读写	0x00000000

## DMA: CH0\_WRITE\_ADDR, CH1\_WRITE\_ADDR, ..., CH14\_WRITE\_ADDR, CH15\_WRITE\_ADDR 寄存器

偏移量：0x004, 0x044, ..., 0x384, 0x3c4

**描述**

DMA通道 N写地址指针

表1149。  
CH0\_WRITE\_ADDR、  
CH1\_WRITE\_ADDR、  
...、CH14\_WRITE\_ADDR  
DR、CH15\_WRITE\_ADDR  
寄存器

位	描述	类型	复位
31:0	该寄存器在每次写操作完成时自动更新。当前值为该通道下一次写入的地址 ◦	读写	0x00000000

## DMA: CH0\_TRANS\_COUNT、CH1\_TRANS\_COUNT、...、CH14\_TRANS\_COUNT、CH15\_TRANS\_COUNT 寄存器

偏移量: 0x008、0x048、...、0x388、0x3c8

**描述**

DMA通道 N传输计数

表1150。  
CH0\_TRANS\_COUNT、  
CH1\_TRANS\_COUNT、...、  
CH14\_TRANS\_COUNT、  
CH15\_TRANS\_COUNT  
寄存器

位	描述	类型	复位
31:28	<p><b>MODE:</b> 当 MODE 为 0x0 时，传输计数随每次传输递减至 0，随后该通道触发 CTRL_CHAIN_TO 指定的下一个通道。</p> <p>当 MODE 为 0x1 时，传输计数随每次传输递减至 0，随后该通道自我重新触发，且同时触发 CTRL_CHAIN_TO 指定的通道。此模式适用于例如带周期性中断的无限循环缓冲 DMA。</p> <p>当 MODE 为 0xf 时，传输计数不会递减。DMA 通道执行无限循环的传输序列，永不触发其他通道或产生中断，直至触发 ABORT。</p> <p>所有其他数值均为保留。</p>	读写	0x0
	枚举值：		
	0x0 → NORMAL		
	0x1 → TRIGGER_SELF		
	0xf → ENDLESS		
27:0	<p><b>COUNT:</b> 28 位传输计数（最多 2.56 亿次传输）。</p> <p>编程设定通道在停止前将执行的总线传输次数。 注意，若传输单位大于一个字节，此数值不等同于传输字节数（详见 CTRL_DATA_SIZE）。</p> <p>通道激活时，读取此寄存器显示剩余传输次数，且每完成一次写传输自动更新。</p> <p>写入此寄存器以设置传输计数器的重载值（RELOAD）。每次触发通道时，重载值（RELOAD）会被复制到当前传输计数器。通道可多次启动，每次均执行最近写入的编程传输次数。</p> <p>RELOAD 值可在 CHx_DBG_TCR 寄存器中读取。如果使用 TRANS_COUNT 作为触发，写入的值将立即作为新传输序列的长度，同时写入 RELOAD 寄存器◦</p>	读写	0x00000000

## DMA: CH0\_CTRL\_TRIG, CH1\_CTRL\_TRIG, ..., CH14\_CTRL\_TRIG, CH15\_C TRL\_TRIG寄存器

偏移量: 0x00c, 0x04c, ..., 0x38c, 0x3cc

### 描述

DMA通道 N控制及状态

表1151。  
CH0\_CTRL\_TRIG,  
CH1\_CTRL\_TRIG, ...  
CH14\_CTRL\_TRIG,  
CH15\_CTRL\_TRIG  
寄存器

位	描述	类型	复位
31	<b>AHB_ERROR:</b> READ_ERROR和WRITE_ERROR标志的逻辑或运算结果。通道在遇到任何总线错误时将停止，并始终置位其通道中断请求（IRQ）标志。	只读	0x0
30	<b>READ_ERROR:</b> 若为1，表示通道发生读总线错误。写1以清除该标志。READ_ADDR 显示发生总线错误的大致地址（不会早于该地址，且不会超过3次传输之后）	WC	0x0
29	<b>WRITE_ERROR:</b> 若为1，表示通道收到写入总线错误。写入1以清除该标志。WRITE_ADDR 显示发生总线错误的大致地址（不会早于该地址，且不会超过5次传输之后）	WC	0x0
28:27	保留。	-	-
26	<b>BUSY:</b> 当通道开始新传输序列时，该标志置高，序列中最后一次传输完成时置低。在 BUSY 置高时清除 EN 会暂停通道，且暂停期间 BUSY 将保持置高状态。  如需提前终止序列（并清除 BUSY 标志），请参见 CHAN_ABORT。	只读	0x0
25	<b>SNIFF_EN:</b> 若为1，则该通道的数据传输对嗅探硬件可见，且每次传输将推进校验和状态。此项仅在启用嗅探硬件且选定该通道时适用。  该机制允许在每个控制块级别启用或禁用校验和。	读写	0x0
24	<b>BSWAP:</b> 对DMA数据应用字节交换转换。 对于字节数据，该操作无效。对于半字数据，每个半字的两个字节将被交换。对于字数据，每个字的四个字节按倒序交换。	读写	0x0
23	<b>IRQ QUIET:</b> 在静默模式下，通道不会在每个传输块结束时产生中断请求（IRQ）。取而代之的是，当向触发寄存器写入NULL时会产生中断请求，表示控制块链的结束。  这减少了在传输多个小控制块的DMA链时CPU需处理的中断数量。	读写	0x0
22:17	<b>TREQ_SEL:</b> 选择传输请求信号。 通道使用传输请求信号来调节数据传输速率。 TREQ信号源包括内部（定时器）或外部（系统的数据请求DREQ）。  0x0至0x3a → 选择DREQ n作为TREQ	读写	0x00
	枚举值：		
	0x3b → TIMER0：选择定时器0作为TREQ		
	0x3c → TIMER1：选择 Timer 1 作为 TREQ		
	0x3d → TIMER2：选择 Timer 2 作为 TREQ（可选）		
	0x3e → TIMER3：选择 Timer 3 作为 TREQ（可选）		

位	描述	类型	复位
	0x3f → PERMANENT：永久请求，适用于无节奏传输。		
16:13	<b>CHAIN_TO</b> : 当该通道完成时，将触发由 CHAIN_TO 指示的通道。通过将 CHAIN_TO 设置为（本通道）来禁用此功能。  注意，该字段会重置为 0，因此通道 1 及以上默认链至通道 0。设置该字段以避免此行为。	读写	0x0
12	<b>RING_SEL</b> : 选择 RING_SIZE 应用于读地址还是写地址。 如果为 0，读地址将在 $(1 \ll \text{RING\_SIZE})$ 边界处回绕。如果为 1，则写入地址回绕。	读写	0x0
11:8	<b>RING_SIZE</b> : 地址回绕区域大小。若为 0，则不回绕。当 $n > 0$ 时，仅地址的低 $n$ 位会发生变化。该操作将在 $(1 \ll n)$ 字节边界上对地址进行环绕，便于访问自然对齐的环形缓冲区。  环形缓冲区的大小可在 2 至 32768 字节之间。此功能可应用于读地址或写地址，具体取决于 RING_SEL 的值。	读写	0x0
	枚举值：		
	0x0 → RING_NONE		
7	<b>INCR_WRITE_REV</b> : 当其值为 1 且 INCR_WRITE 为 1 时，写地址会在每次传输时递减，而非递增。  当其值为 1 且 INCR_WRITE 为 0 时，此未使用的组合会使写地址每次递增两倍传输大小，即跳过交替地址。	读写	0x0
6	<b>INCR_WRITE</b> : 当其值为 1 时，写地址会在每次传输时递增。当其值为 0 时，每次写入均指向相同的初始地址。  通常，应在内存到外设传输中禁用此功能。	读写	0x0
5	<b>INCR_READ_REV</b> : 当其值为 1 且 INCR_READ 为 1 时，读地址会在每次传输时递减，而非递增。  如果为 1 且 INCR_READ 为 0，则此原本未使用的组合会导致读取地址增加两倍传输大小，即跳过交替地址。	读写	0x0
4	<b>INCR_READ</b> : 如果为 1，读取地址在每次传输时递增。如果为 0，则每次读取均指向相同的初始地址。  通常应为外设到内存的传输禁用此选项。	读写	0x0
3:2	<b>DATA_SIZE</b> : 设置每次总线传输的数据大小（字节/半字/字）。 READ_ADDR 和 WRITE_ADDR 在每次传输时按此大小（1/2/4 字节）递增。	读写	0x0
	枚举值：		
	0x0 → SIZE_BYTE		
	0x1 → SIZE_HALFWORD		
	0x2 → SIZE_WORD		

位	描述	类型	复位
1	<p><b>HIGH_PRIORITY:</b> HIGH_PRIORITY 赋予通道在调度发起时优先权：在每轮调度中，所有高优先级通道会被优先考虑，然后仅选择一个低优先级通道，之后返回高优先级通道。</p> <p>这仅影响 DMA 调度通道的顺序。DMA 的总线优先级未发生变化。如果 DMA 未达到饱和状态，则低优先级通道不会出现吞吐量损失。</p>	读写	0x0
0	<p><b>EN:</b> DMA 通道使能。 当值为 1 时，通道将响应触发事件，从而使其变为 BUSY 状态并开始传输数据。当值为 0 时，通道将忽略触发信号，停止发起传输，并暂停当前传输序列（即若 BUSY 已为高电平，仍保持高电平）。</p>	读写	0x0

## DMA: CH0\_AL1\_CTRL、CH1\_AL1\_CTRL、.....、CH14\_AL1\_CTRL、CH15\_AL1\_CTRL 寄存器

偏移量：0x010、0x050、.....、0x390、0x3d0

表 1152。  
CH0\_AL1\_CTRL  
' CH1\_AL1\_CTRL  
' .....、CH14\_AL1\_CTRL、CH15\_AL1\_CTRL 寄存器

位	描述	类型	复位
31:0	通道 NCTRL 寄存器别名	读写	-

## DMA: CH0\_AL1\_READ\_ADDR, CH1\_AL1\_READ\_ADDR, ..., CH14\_AL1\_READ\_ADDR, CH15\_AL1\_READ\_ADDR 寄存器

偏移量：0x014, 0x054, ..., 0x394, 0x3d4

表 1153。  
CH0\_AL1\_READ\_ADDR  
' CH1\_AL1\_READ\_ADDR  
' .....、CH14\_AL1\_READ\_ADDR  
' R, CH15\_AL1\_READ\_ADDR  
' R 寄存器

位	描述	类型	复位
31:0	通道 NREAD_ADDR 寄存器别名	读写	-

## DMA: CH0\_AL1\_WRITE\_ADDR, CH1\_AL1\_WRITE\_ADDR, ..., CH14\_AL1\_WRITE\_ADDR, CH15\_AL1\_WRITE\_ADDR 寄存器

偏移量：0x018, 0x058, ..., 0x398, 0x3d8

表 1154。  
CH0\_AL1\_WRITE\_ADDR  
' R, CH1\_AL1\_WRITE\_ADDR  
' R, .....、CH14\_AL1\_WRITE\_ADDR  
' DR, CH15\_AL1\_WRITE\_ADDR  
' DR 寄存器

位	描述	类型	复位
31:0	通道 NWRITE_ADDR 寄存器别名	读写	-

## DMA: CH0\_AL1\_TRANS\_COUNT\_TRIG, CH1\_AL1\_TRANS\_COUNT\_TRIG, ..., CH14\_AL1\_TRANS\_COUNT\_TRIG, CH15\_AL1\_TRANS\_COUNT\_TRIG 寄存器

偏移：0x01c, 0x05c, ..., 0x39c, 0x3dc

表 1155。  
CH0\_AL1\_TRANS\_CO\_NT\_TRIG, CH1\_AL1\_TRANS\_CO\_NT\_TRIG, .....、CH14\_AL1\_TRANS\_CO\_NT\_TRIG, CH15\_AL1\_TRANS\_COUNT\_TRIG 寄存器

位	描述	类型	复位
31:0	<p>通道 NTRANS_COUNT 寄存器的别名</p> <p>该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。</p>	读写	-

## DMA: CH0\_AL2\_CTRL, CH1\_AL2\_CTRL, ..., CH14\_AL2\_CTRL, CH15\_AL2\_CTRL 寄存器

偏移: 0x020, 0x060, ..., 0x3a0, 0x3e0

表1156。  
CH0\_AL2\_CTRL,  
CH1\_AL2\_CTRL, ...  
, CH14\_AL2\_CTRL  
, CH15\_AL2\_CTRL  
L寄存器

位	描述	类型	复位
31:0	通道 NCTRL 寄存器别名	读写	-

### DMA: CH0\_AL2\_TRANS\_COUNT, CH1\_AL2\_TRANS\_COUNT, ..., CH14\_AL2\_TRANS\_COUNT, CH15\_AL2\_TRANS\_COUNT寄存器

偏移量: 0x024, 0x064, ..., 0x3a4, 0x3e4

表1157。  
CH0\_AL2\_TRANS\_CO  
UNT、CH1\_AL2  
\_TRANS\_COUNT、...  
, CH14\_AL2\_T  
RANS\_COUNT、CH15  
\_AL2\_TRANS\_C  
OUNT寄存器

位	描述	类型	复位
31:0	通道 NTRANS_COUNT寄存器的别名	读写	-

**DMA: CH0\_AL2\_READ\_ADDR, CH1\_AL2\_READ\_ADDR, ..., CH14\_AL2\_READ\_ADDR、CH15\_AL2\_READ\_ADDR寄存器**

偏移量: 0x028, 0x068, ..., 0x3a8, 0x3e8

表1158。  
CH0\_AL2\_READ\_ADDR  
, CH1\_AL2\_READ\_ADDR  
..., CH14\_AL  
2\_READ\_ADDR, CH1  
5\_AL2\_READ  
\_ADDR寄存器

位	描述	类型	复位
31:0	通道 NREAD_ADDR 寄存器别名	读写	-

**DMA: CH0\_AL2\_WRITE\_ADDR\_TRIG, CH1\_AL2\_WRITE\_ADDR\_TRIG, ..., CH14\_AL2\_WRITE\_ADDR\_TRIG, CH15\_AL2\_WRITE\_ADDR\_TRIG寄存器**

偏移量: 0x02c, 0x06c, ..., 0x3ac, 0x3ec

表1159。  
CH0\_AL2\_WRITE\_ADD  
R\_TRIG,  
CH1\_AL2\_WRITE\_ADD  
R\_TRIG, ...  
CH14\_AL2\_WRITE\_AD  
DR\_TRIG,  
CH15\_AL2\_WRITE\_AD  
DR\_TRIG寄存器

位	描述	类型	复位
31:0	通道 NWRITE_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道 计数器并启动该通道。	读写	-

**DMA: CH0\_AL3\_CTRL, CH1\_AL3\_CTRL, ..., CH14\_AL3\_CTRL, CH15\_AL3\_CTRL寄存器**

偏移量: 0x030, 0x070, ..., 0x3b0, 0x3f0

表1160。  
CH0\_AL3\_CTRL,  
CH1\_AL3\_CTRL, ...  
CH14\_AL3\_CTRL,  
CH15\_AL3\_CTRL  
寄存器

位	描述	类型	复位
31:0	通道 NCTRL 寄存器别名	读写	-

**DMA: CH0\_AL3\_WRITE\_ADDR, CH1\_AL3\_WRITE\_ADDR, ..., CH14\_AL3\_WRITE\_ADDR, CH15\_AL3\_WRITE\_ADDR 寄存器**

偏移量: 0x034, 0x074, ..., 0x3b4, 0x3f4

表1161。  
CH0\_AL3\_WRITE\_ADD  
R,  
CH1\_AL3\_WRITE\_ADD  
R, ...  
CH14\_AL3\_WRITE\_AD  
DR,  
CH15\_AL3\_WRITE\_AD  
DR寄存器

位	描述	类型	复位
31:0	通道 NWRITE_ADDR 寄存器别名	读写	-

**DMA: CH0\_AL3\_TRANS\_COUNT, CH1\_AL3\_TRANS\_COUNT, ..., CH14\_AL3\_TRANS\_COUNT, CH15\_AL3\_TRANS\_COUNT寄存器**

偏移量: 0x038、0x078、...、0x3b8、0x3f8

表1162。  
`CH0_AL3_TRANS_CO  
NT,`  
`CH1_AL3_TRANS_CO  
NT, ...`  
`CH14_AL3_TRANS_CO  
UNT,`  
`CH15_AL3_TRANS_CO  
UNT`寄存器

位	描述	类型	复位
31:0	通道 NTRANS_COUNT 寄存器的别名	读写	-

## DMA: CH0\_AL3\_READ\_ADDR\_TRIG, CH1\_AL3\_READ\_ADDR\_TRIG, ..., CH14\_AL3\_READ\_ADDR\_TRIG, CH15\_AL3\_READ\_ADDR\_TRIG 寄存器

偏移: 0x03c, 0x07c, ..., 0x3bc, 0x3fc

表1163。  
`CH0_AL3_READ_ADDR  
_TRIG,`  
`CH1_AL3_READ_ADDR  
_TRIG, ...`  
`CH14_AL3_READ_ADD  
R_TRIG,`  
`CH15_AL3_READ_ADD  
R_TRIG`寄存器

位	描述	类型	复位
31:0	通道 NREAD_ADDR 寄存器别名 该寄存器为触发寄存器 (0xc)。写入非零值将重载通道计数器并启动该通道。	读写	-

## DMA: INTR 寄存器

偏移: 0x400

### 描述

中断状态 (原始)

表1164. INTR  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	DMA通道0至15的原始中断状态。位n对应通道n。 忽略所有屏蔽或强制操作。通道中断可通过向INTR或INTS0/1/2/3写入位掩码来清除。  通道中断可根据INTE0、INTE1、INTE2和INTE3的配置，路由至四个系统级IRQ中的任意一个。  多路系统级中断可用于允许NVIC IRQ抢占更关键的通道，分散IRQ负载至不同内核，或将IRQ定向至不同的安全域。  忽略多个IRQ，仅使用INTE0/INTS0/IRQ 0，同样有效。  如果以低于某通道安全/权限级别（由该通道的SECCFG_CHx寄存器定义）的安全/权限级别访问此寄存器，则该通道的中断状态将读取为0，写入操作将被忽略。	WC	0x0000

## DMA: INT0 寄存器

偏移: 0x404

### 描述

IRQ0 中断使能

表1165. INTFO寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	置位第n位以将通道n的中断传递至DMA IRQ 0。  注意：若通道安全/权限级别（由SECCFG_CHx定义）高于IRQ安全/权限级别（由SECCFG_IRQ0定义），该位无效。	读写	0x0000

## DMA：INTFO寄存器

偏移: 0x408

### 描述

强制中断

表1166. INTFO寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	写入1以强制对应INTSO中的位。中断将保持断言状态，直至清除INTFO 。  在此寄存器中，安全/权限级别（SECCFG_CHx）高于 SECCFG_IRQ0 的通道读作0，且写入操作将被忽略。	读写	0x0000

## DMA：INTSO 寄存器

偏移: 0x40c

### 描述

IRQ 0 的中断状态

表 1167. INTSO 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	指示当前导致 IRQ 0 被触发的活动通道中断请求。  可通过在此处写入位掩码来清除通道中断。  在此寄存器中，安全/权限级别（SECCFG_CHx）高于 SECCFG_IRQ0 的通道读作0，且写入操作将被忽略。	WC	0x0000

## DMA：INTE1 寄存器

偏移: 0x414

### 描述

IRQ 1 的中断使能

表 1168. INTE1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	设置位 n 以将来自通道 n 的中断传递至 DMA IRQ 1。  注意：若通道的安全/权限级别（由 SECCFG_CHx 定义）高于 IRQ 的安全/权限级别（由 SECCFG_IRQ1 定义），该位无效。	读写	0x0000

## DMA：INTF1 寄存器

偏移: 0x418

### 描述

强制中断

表 1169. INTF1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	写入1以强制将对应位设置至INTS1。中断保持有效，直至INTF1被清除。 。	读写	0x0000

## DMA：INTS1寄存器

偏移: 0x41c

### 描述

IRQ 1 的中断状态

表1170. INTS1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	指示当前触发IRQ 1断言的活动通道中断请求。  可通过在此处写入位掩码来清除通道中断。  安全/特权级别（SECCFG_CHx）高于SECCFG_IRQ1的通道在本寄存器中读作0，且忽略写入。	WC	0x0000

## DMA：INTE2寄存器

偏移: 0x424

### 说明

IRQ 2 的中断使能

表1171. INTE2  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	置位bit n以允许通道n的中断传递至DMA IRQ 2。  注意，若通道安全/特权级别（由SECCFG_CHx定义）高于IRQ安全/特权级别（由SECCFG_IRQ2定义），该位无效。	读写	0x0000

## DMA：INTF2寄存器

偏移: 0x428

### 说明

强制中断

表1172. INTF2  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	写入1以强制INTS2中对应的位。中断将持续维持，直至清除INTF2。	读写	0x0000

## DMA：INTS2寄存器

偏移: 0x42c

### 描述

IRQ 2 的中断状态

表1173. INTS2  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	指示当前导致IRQ 2触发的活动通道中断请求。  可通过在此处写入位掩码来清除通道中断。  安全/权限级别（SECCFG_CHx）高于SECCFG_IRQ2的通道，在此寄存器中读作0，且写入无效。	WC	0x0000

## DMA：INTE3寄存器

偏移: 0x434

### 描述

IRQ 3 的中断使能

表1174. INTE3  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	设置位n以允许来自通道n的中断通过至DMA IRQ 3。  注意：当通道安全/权限级别（由SECCFG_CHx定义）高于IRQ安全/权限级别（由SECCFG_IRQ3定义）时，该位无效。	读写	0x0000

## DMA：INTF3寄存器

偏移: 0x438

### 描述

强制中断

表1175. INTF3  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	写入1以强制INTS3中相应位。中断将保持激活状态，直到清除INTF3。	读写	0x0000

## DMA：INTS3寄存器

偏移: 0x43c

### 描述

IRQ 3 的中断状态

表1176. INTS3  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	指示当前导致IRQ 3断言的活动通道中断请求。  可通过在此处写入位掩码来清除通道中断。  在本寄存器中，安全/权限(SECCFG_CHx)高于SECCFG_IRQ3的通道读取值恒为0，且忽略写入操作。	WC	0x0000

## DMA：TIMER0、TIMER1、TIMER2、TIMER3寄存器

偏移: 0x440, 0x444, 0x448, 0x44c

### 描述

节奏 (X/Y) 分数定时器

节奏定时器以 $((X/Y) \times \text{sys\_clk})$ 的速率产生TREQ断言。该公式每个sys\_clk周期计算一次，因此只能生成速率为每个sys\_clk周期不超过1次（即持续TREQ）或更低的TREQ。

表1177. TIMER0,  
TIMER1, TIMER2,  
TIMER3 寄存器

位	描述	类型	复位
31:16	X：步进定时器被除数。指定 (X/Y) 分数定时器中的 X 值。	读写	0x0000
15:0	Y：步进定时器除数。指定 (X/Y) 分数定时器中的 Y 值。	读写	0x0000

## DMA：MULTI\_CHAN\_TRIGGER 寄存器

偏移: 0x450

**描述**

同时触发一个或多个通道

表1178。  
MULTI\_CHAN\_TRIGGER  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	该寄存器的每个位对应一个 DMA 通道。向相关位写入 1 等同于写入该通道的触发寄存器；只要该通道当前已启用且未占用，通道即会启动。	SC	0x0000

**DMA：SNIFF\_CTRL 寄存器**

偏移：0x454

**说明**

嗅探器控制

表1179。  
SNIFF\_CTRL 寄存器

位	描述	类型	复位
31:12	保留。	-	-
11	<b>OUT_INV</b> ：若设置，读取结果将以位反码形式显示。 这不影响校验和的计算方式；结果为在结果寄存器与总线之间实时转换。	读写	0x0
10	<b>OUT_REV</b> ：若设置，读取时结果以位反转形式显示。此操作不影响校验和的计算方式；结果在结果寄存器与总线之间实时转换。	读写	0x0
9	<b>BSWAP</b> ：在传入校验和计算之前，对嗅探数据执行局部字节反转。  请注意，嗅探硬件位于read master中DMA通道字节交换之后：若通道CTR_L_BSWAP和SNIFF_CTRL_BSWAP均启用，嗅探器视角下二者效果相互抵消。	读写	0x0
8:5	<b>CALC</b>  枚举值：  0x0 → CRC32：计算CRC-32（IEEE802.3多项式） 0x1 → CRC32R：计算带位反转数据的CRC-32（IEEE802.3多项式） 0x2 → CRC16：计算CRC-16-CCITT 0x3 → CRC16R：计算带位反转数据的CRC-16-CCITT 0xe → EVEN：对所有数据执行 XOR 归约。== 1 表示总的 1 的数量为奇数。 0xf → SUM：计算简单的 32 位校验和（采用 32 位累加器进行加法）。	读写	0x0
4:1	<b>DMACH</b> ：用于监听的 DMA 通道	读写	0x0
0	<b>EN</b> ：启用监听器	读写	0x0

**DMA：SNIFF\_DATA 寄存器**

Offset: 0x458

**描述**

用于嗅探硬件的数据累加器

表 1180。  
SNIFF\_DATA 寄存器

位	描述	类型	复位
31:0	在开始对 SNIFF_CTRL_DMACH 指定的通道进行 DMA 传输前, 请在此写入初始种子值。每当硬件检测到对指定通道的读取时, 都会更新该寄存器。通道完成后, 可从该寄存器读取最终结果。	读写	0x00000000

**DMA: FIFO\_LEVELS 寄存器**

Offset: 0x460

**描述**

调试 RAF、WAF、TDF 级别

表 1181。  
FIFO\_LEVELS 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>RAF_LVL</b> : 当前读地址 FIFO 的填充级别	只读	0x00
15:8	<b>WAF_LVL</b> : 当前写地址 FIFO 的填充级别	只读	0x00
7:0	<b>TDF_LVL</b> : 当前传输数据 FIFO 填充级别	只读	0x00

**DMA: CHAN\_ABORT 寄存器**

偏移: 0x464

**描述**

中止一个或多个通道上正在进行的传输序列

表 1182。  
CHAN\_ABORT  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	每个位对应一个通道。写入1将中止该通道上正在进行的传输序列。该位将保持高电平, 直到所有正在传输的数据通过地址和数据 FIFO 全部清空。  写入后, 必须轮询该寄存器, 直到其返回全零值。在此之前, 重新启动该通道是不安全的。	SC	0x0000

**DMA: N\_CHANNELS 寄存器**

偏移: 0x468

表 1183。  
N\_CHANNELS 寄存器

位	描述	类型	复位
31:5	保留。	-	-
4:0	该 DMA 实例配备的通道数量。此 DMA 支持最多 16 个硬件通道, 但可配置为仅使用一个, 以最大限度减少硅片面积。	只读	-

**DMA: SECCFG\_CH0、SECCFG\_CH1、...、SECCFG\_CH14、SECCFG\_CH15  
寄存器**

偏移: 0x480、0x484、...、0x4b8、0x4bc

**描述**

信道 N 的安全配置。控制该信道是否执行安全/非安全及特权/非特权总线访问。

若该信道产生某一安全级别的总线访问，则在编程、触发、中止、状态检查、中断响应或中断确认时，必须至少具备该级别的访问权限（级别顺序为 S+P > S+U > NS+P > NS+U）。

一旦软件开始配置该信道，该寄存器将自动锁定（变为只读）。

该寄存器为全局可读，但仅允许在安全且具特权的上下文中写入。

表 1184。  
SECCFG\_CH0,  
SECCFG\_CH1, ...,  
SECCFG\_CH14,  
SECCFG\_CH15  
寄存器

位	描述	类型	复位
31:3	保留。	-	-
2	<b>LOCK</b> : 复位时 LOCK 为 0，且在成功写入该信道控制寄存器后自动置为 1。即对 CTRL、READ_ADDR、WRITE_ADDR、TRANS_COUNT 及其别名的写入。  一旦LOCK位被置位，该寄存器即变为只读。  例如，若写操作的权限低于通道SECCFG寄存器中规定的权限而导致写入失败，则不会设置LOCK位。	读写	0x0
1	<b>S</b> : 安全通道。当为1时，该通道执行安全总线访问。当为0时，该通道执行非安全总线访问。  当为1时，该通道仅能在安全上下文中被控制。	读写	0x1
0	<b>P</b> : 特权通道。当为1时，该通道执行特权总线访问。当为0时，该通道执行无特权总线访问。  当为1时，该通道仅能由同一安全/非安全级别的特权上下文，或任何更高级安全/非安全级别的上下文进行控制。	读写	0x1

**DMA: SECCFG\_IRQ0、SECCFG\_IRQ1、SECCFG\_IRQ2、SECCFG\_IRQ3 寄存器**

偏移量：0x4c0、0x4c4、0x4c8、0x4cc

**描述**

IRQ N 的安全配置。控制IRQ是否允许被非安全或非特权上下文配置，以及是否允许其监视安全或特权通道的中断标志。

表 1185。  
SECCFG\_IRQ0  
、SECCFG\_IRQ1、SECCFG\_IRQ2、  
SECCFG\_IRQ3  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>S</b> : 安全 IRQ。若为 1，则该 IRQ 的控制寄存器仅能从安全上下文访问。  若为 0，则该 IRQ 的控制寄存器可从非安全上下文访问，但根据 SECCFG_C_Hx，安全通道在 IRQ 状态中被屏蔽，且该 IRQ 的寄存器不能用于确认安全通道的中断。	读写	0x1

位	描述	类型	复位
0	<p><b>P</b>: 特权 IRQ。若为 1，则该 IRQ 的控制寄存器仅能从特权上下文访问。</p> <p>若为 0，则该 IRQ 的控制寄存器可从非特权上下文访问，但根据 SECCFG_CHx，特权通道在 IRQ 状态中被屏蔽，且该 IRQ 的寄存器不能用于确认特权通道的中断。</p>	读写	0x1

## DMA: SECCFG\_MISC 寄存器

偏移: 0x4d0

### 描述

其它安全配置

表1186。  
SECCFG\_MISC  
寄存器

位	描述	类型	复位
31:10	保留。	-	-
9	<b>TIMER3_S</b> : 若值为1，TIMER3寄存器仅能从安全上下文访问，且定时器DREQ 3仅对安全通道可见。	读写	0x1
8	<b>TIMER3_P</b> : 若值为1，TIMER3寄存器仅能从特权（或更安全）上下文访问，且定时器DREQ 3仅对特权（或更安全）通道可见。	读写	0x1
7	<b>TIMER2_S</b> : 若值为1，TIMER2寄存器仅能从安全上下文访问，且定时器DREQ 2仅对安全通道可见。	读写	0x1
6	<b>TIMER2_P</b> : 若值为1，TIMER2寄存器仅能从特权（或更安全）上下文访问，且定时器DREQ 2仅对特权（或更安全）通道可见。	读写	0x1
5	<b>TIMER1_S</b> : 若值为1，TIMER1寄存器仅能从安全上下文访问，且定时器DREQ 1仅对安全通道可见。	读写	0x1
4	<b>TIMER1_P</b> : 如果为1，则TIMER1寄存器仅允许从特权（或更高安全级别）上下文访问，且计时器DREQ 1仅对特权（或更高安全级别）通道可见。	读写	0x1
3	<b>TIMER0_S</b> : 如果为1，则TIMER0寄存器仅允许从安全上下文访问，且计时器DREQ 0仅对安全通道可见。	读写	0x1
2	<b>TIMER0_P</b> : 如果为1，则TIMER0寄存器仅允许从特权（或更高安全级别）上下文访问，且计时器DREQ 0仅对特权（或更高安全级别）通道可见。	读写	0x1
1	<p><b>SNIFF_S</b>: 如果为1，嗅探器可以查看来自安全通道的数据传输，且嗅探器本身仅允许从安全上下文访问。</p> <p>如果为0，嗅探器允许从安全或非安全上下文访问，但无法查看安全通道的数据传输。</p>	读写	0x1
0	<p><b>SNIFF_P</b>: 如果为1，嗅探器可以查看来自特权通道的数据传输，且嗅探器本身仅允许从特权上下文访问，或在SNIFF_S为0时允许从安全上下文访问。</p> <p>若为0，嗅探器可从特权或非特权上下文（具备足够安全级别）访问，但无法查看来自特权通道的传输。</p>	读写	0x1

## DMA: MPU\_CTRL 寄存器

偏移: 0x500

### 描述

DMA MPU 控制寄存器，仅可在特权模式下访问。

表 1187。  
MPU\_CTRL 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>NS_HIDE_ADDR</b> : 默认情况下，当某区域的S位为0时，非安全特权读取可见该区域的基地址和限制地址。设置此位后，即便该区域为非安全，非安全读取时地址亦显示为0，以防泄露处理器SAU映射信息。	读写	0x0
2	<b>S</b> : 判断未被活动MPU区域覆盖的地址为安全 (1) 或非安全 (0)	读写	0x0
1	<b>P</b> : 判断未被活动MPU区域覆盖的地址为特权 (1) 或非特权 (0)	读写	0x0
0	保留。	-	-

## DMA: MPU\_BAR0、MPU\_BAR1、.....、MPU\_BAR6、MPU\_BAR7 寄存器

偏移: 0x504, 0x50c, ..., 0x534, 0x53c

### 描述

MPU 区域 N 的基地址寄存器。仅可在安全且特权模式下写入。

表 1188。  
MPU\_BAR0,  
MPU\_BAR1, ...  
MPU\_BAR6,  
MPU\_BAR7 寄存器

位	描述	类型	复位
31:5	<b>ADDR</b> : 本 MPU 区域匹配的地址为 addr[31:5] (最高27位) 大于等于 BAR_A DDR 且小于等于 LAR_ADDR。  仅当该区域的 S 位清零且 MPU_CTRL_NS_HIDE_ADDR 清零时，可从任何特权上下文读取。否则，仅能从安全特权上下文读取。	读写	0x0000000
4:0	保留。	-	-

## DMA: MPU\_LAR0, MPU\_LAR1, ..., MPU\_LAR6, MPU\_LAR7 寄存器

偏移: 0x508, 0x510, ..., 0x538, 0x540

### 描述

MPU 区域 N 的限制地址寄存器。除 P 位外，仅可在安全且特权模式下写入。

表 1189。  
MPU\_LAR0, MPU\_LAR1, ..., MPU\_LAR6, MPU\_LAR7  
寄存器

位	描述	类型	复位
31:5	<b>ADDR</b> : 限制地址位 31:5。仅当此区域的 S 位清除且 MPU_CTRL_NS_HIDE_ADDR 清除时，方可从任何特权上下文读取。 否则，仅能从安全特权上下文读取。	读写	0x0000000
4:3	保留。	-	-

位	描述	类型	复位
2	<b>S</b> : 若启用此区域，确定匹配该区域地址的安全/非安全 (=1/0) 状态。	读写	0x0
1	<b>P</b> : 若启用此区域，确定匹配该区域地址的特权/非特权 (=1/0) 状态。仅当 S 位清除时，方可从任何特权上下文写入。否则，仅能从安全的特权上下文写入。	读写	0x0
0	<b>EN</b> : 区域使能。若为 1，则基地址 (BAR_ADDR) 与限制地址 (LAR_A DDR) 指定范围内的任一地址均具有由 S 和 P 指定的属性。	读写	0x0

## DMA:CH0\_DBG\_CTDREQ, CH1\_DBG\_CTDREQ, ..., CH14\_DBG\_CTDREQ, CH15\_DBG\_CTDREQ 寄存器

偏移量: 0x800, 0x840, ..., 0xb80, 0xbc0

表1190。  
CH0\_DBG\_CTDREQ,  
CH1\_DBG\_CTDREQ, ...  
CH14\_DBG\_CTDREQ,  
CH15\_DBG\_CTDREQ  
寄存器

位	描述	类型	复位
31:6	保留。	-	-
5:0	读取：获取通道DREQ计数器（即DMA预计可对外设执行的无溢出/欠流访问次数）。写入任意值时：清除计数器，并使通道重新初始化DREQ握手。	WC	0x00

## DMA:CH0\_DBG\_TCR、CH1\_DBG\_TCR、...、CH14\_DBG\_TCR、CH15\_DBG\_TCR 寄存器

偏移量: 0x804, 0x844, ..., 0xb84, 0xbc4

表1191。  
CH0\_DBG\_TCR  
、CH1\_DBG\_TCR  
、...、CH14\_DB  
G\_TCR、CH15\_  
DBG\_TCR寄存器

位	描述	类型	复位
31:0	读取以获取通道TRANS_COUNT的重载值，即下一次传输的长度	只读	0x00000000

## 12.7. USB

### 12.7.1. 概述

#### 注意

所需的先决知识

本节要求具备USB协议相关知识。若您尚未熟悉USB协议，建议参考非常实用的“USB简单入门”网站存档。有关本节术语的正式定义，请参阅USB 2.0规范。

RP2350包含一款USB 2.0控制器，能够以下列任一方方式运行：

- 全速 (FS) 设备 (12 Mb/s)
- 主机，可与低速 (LS) (1.5 Mb/s) 及全速设备通信，包括连接于USB集线器的多个下游设备

集成USB 1.1 PHY，用以连接USB控制器与芯片的DP和DM引脚。当USB控制器未使用时，可将其用作3.3 V GPIO。

### 12.7.1.1. 功能特性

USB控制器硬件负责处理底层USB协议。程序员必须配置控制器、提供数据缓冲区，并根据总线事件消费或提供数据缓冲区。控制器在需要处理时会中断处理器。USB 控制器配备了 4 kB 双端口 SRAM (DPSRAM)，用于配置和数据缓冲。

#### 12.7.1.1.1. 设备模式

在设备模式下，USB 控制器具有以下特性：

- 兼容 USB 2.0 的全速设备 (12 Mb/s)
- 支持最多 32 个端点 (端点 0 → 15，包含输入和输出方向)
- 支持控制、同步 (ISO)、批量和中断端点类型
- 支持双缓冲
- DPSRAM 中有 3840 字节可用缓冲空间，相当于 60 个 64 字节缓冲区

#### 12.7.1.1.2. 主机模式

在主机模式下，USB 控制器能够：

- 与全速 (12 Mb/s) 设备和低速设备 (1.5 Mb/s) 通信
- 通过 USB 集线器与多个设备通信，包括连接到全速集线器的低速设备
- 硬件轮询最多 15 个中断端点 (用于集线器通知主机连接/断开事件，鼠标通知主机移动等)

#### 12.7.1.1.3. USB DPRAM

USB 控制器使用 4 kB 的双端口 SRAM (DPSRAM) 与控制器交换数据及控制信息。这也称为双端口 RAM (DPRAM)。一个端口可通过系统总线访问，由 `clk_sys` 时钟驱动。另一个端口可由控制器访问，由 `clk_usb` 时钟驱动。DPRAM 映射于系统地址空间，起始地址为 `0x50100000`, `USBCTRL_DPRAM_BASE`。

USB DPRAM 支持 32 位、16 位及 8 位的读写操作。写操作在一个时钟周期内完成。读操作在两个时钟周期内完成。

您可在 USB DPRAM 中存储不用于 USB 控制器操作的一般用户数据。当控制器被禁用时，4 kB 的 DPRAM 全部可用。访问 DPRAM 之前，必须先使 USB 控制器退出复位状态。

由于USB控制器位于外设地址空间，处理器无法通过指令读取该区域。

无论处理器的SAU/MPU/PMP或系统ACCESSCTRL寄存器如何配置，尝试从USB DPRAM读取指令均无条件返回总线错误响应。

由于外设地址在IDAU（第10.2.2节）中标记为豁免，故该地址范围的SAU配置将被忽略。对USB DPRAM的访问仅由处理器的MPU/PMP及ACCESSCTRL的USBCTRL寄存器控制。

### 12.7.2. RP2040的变更内容

RP2040的所有更改均为RP2040功能集的超集。RP2040 USB控制器的既有软件仍然兼容，但唯一例外是：必须在启动及断电事件后清除MAIN\_CTRL.PHY\_ISO位。建议将LINESTATE\_TUNING寄存器保持其复位值不变。软件不得清除此寄存器。

### 12.7.2.1. 勘误修正

RP2350修正了所有RP2040 USB相关的勘误。本修复包含了对以下RP2040B0及B1错误的修正，这些错误同样已由RP2040B2解决：

- RP2040-E2：USB设备端点中止信号未被清除
- RP2040-E5：USB设备在繁忙的USB总线上无法退出复位状态

有关RP2040B2的详细信息，请参阅RP2040数据手册。

RP2350修正了以下RP2040B2错误，这些错误在RP2040B2上需通过软件进行解决：

- RP2040-E3：USB主机模式下，中断端点缓冲结束标志可能因缓冲选择错误而被错误置位
- RP2040-E4：USB主机在单缓冲模式下错误写入缓冲区状态的上半部分
- RP2040-E15：在IN传输过程中发生特定总线错误时，USB设备控制器可能挂起（详见第12.7.2.4节）

### 12.7.2.2 新特性

#### 12.7.2.2.1 概述

- USB PHY 的 DP 和 DM 引脚可用作普通 GPIO 引脚。详见第9.4节中的GPIO复用表646。
- MAIN\_CTRL.PHY\_ISO 控制用于在切换核心电源域断电时，将 PHY 与该电源域隔离。隔离控制复位为1，意味着在使用PHY之前必须清除 MAIN\_CTRL.PHY\_ISO 位。有关隔离的详细信息，请参见第9章。
- SIE\_CTRL.PULLDOWN\_EN 默认值为1，以匹配USB PHY中隔离锁存器的复位状态。将DP和DM引脚默认下拉，可防止未使用时引脚悬空，从而节省能耗。
- USB\_MUXING.TO\_PHY 位默认值为1，以匹配隔离锁存器的复位状态。
- 新增 SM\_STATE，用于公开控制器各模块的内部状态。

#### 12.7.2.2.2. 主机

- 现在可选择在接收到 NAK 时停止事务。此功能允许USB主机在设备无法传输数据时停止批量传输。某些使用批量端点的设备，例如UART，会持续返回 NAK，直至接收到字符。通过硬件停止传输而非采用软件方式，主机可接收到 NAK，确保无数据丢失。RP2350新增了两个寄存器位及一个中断以支持该功能：
  - NAK\_POLL.STOP\_EPX\_ON\_NAK 控制位，用于启用或禁用此功能。
  - NAK\_POLL.EPX\_STOPPED\_ON\_NAK 状态位，亦配有相应中断 INTS.EPX\_STOPPED\_ON\_NAK。
- RP2350延长了包间及转换超时，以适应最恶劣的集线器延迟。该问题仅在长链USB集线器中出现，但实际应用中未曾观察到。主机状态机时序已修正以符合USB规范。此修正通过LINESTATE\_TUNING.MULTI\_HUB\_FIX启用。

#### 12.7.2.2.3. 设备

- 添加了从挂起状态唤醒的修正：任何总线活动（定义为 K 或 SE0）都应触发从挂起状态唤醒，而不仅限于合格的恢复信号周期。该修正默认启用，可通过LINESTATE\_TUNING.DEV\_LS\_WAKE\_FIX禁用（此处的LS表示线路状态，而非低速）。
- 新增DPSRAM双重读取功能以确保数据一致性。此举避免了在缓冲区控制寄存器中需单独设置 AVAILABLE 位以外的缓冲区信息。该功能默认启用，由LINESTATE\_TUNING.DEV\_BUFF\_CONTROL\_DOUBLE\_READ\_FIX控制。

- 新增短包接收时停止设备从主机传出的功能。对于 EP0，该功能由 SIE\_CTRL.EP0\_STOP\_ON\_SHORT\_PACKET 控制，方法是停止事务并在双缓冲模式下不切换缓冲区。还新增 short\_packet 中断，用以通知软件已收到短包 (INTS.RX\_SHORT\_PACKET)。

#### 12.7.2.4. 设备错误处理

- 新增 DEV\_RX\_ERR QUIESCE 功能：设备端点错误计数会复制主机内部的 Cerr 计数，从而使软件能够检测主机是否在连续三次错误后可能已停止该端点。RX 解码的各个阶段会生成各自的错误信号，并传播至顶层。这些错误信号在不同时间到达，因此每次传输失败将产生两个错误中断。新增该行为的可选覆盖功能，通过在传输过程中首次发生错误时强制设备 RX 控制器进入空闲状态实现。该修复通过 LINESTATE\_TUNING.DEV\_RX\_ERR QUIESCE 启用。
- 新增 SIE\_RX\_CHATTER\_SE0\_FIX：现有错误恢复机制在发出帧错误信号并返回空闲状态前需等待 8 个 FS 空闲位时间。此方法对随机总线错误有效，但当集线器终止下行数据包时，集线器会强制产生位填充错误，随后是结束包 (EOP)。主机可能立即发送有效令牌，但设备控制器可能因该强制延迟而忽略该令牌。可选择等待有效的 EOP 或 8 个空闲位时间，然后发出帧错误信号。要启用该修复，请使用 LINESTATE\_TUNING.SIE\_RX\_CHATTER\_SE0\_FIX。
- 修复 RP2040-E15：接收状态机并非总能正确处理位流解串器中断传输的情况。若在数据包中期因位填充错误导致解码终止，设备控制器可能会锁死。若解串器标记了位填充错误并随后在位状态返回空闲后发出帧错误信号，则无条件禁用 RX。要启用该修复，请使用 LINESTATE\_TUNING.SIE\_RX\_BITSTUFF\_FIX。
- 设备状态机看门狗：新增看门狗功能，若设备状态机卡住超过设定时间，可强制其进入空闲状态。此项用于处理上述修复未覆盖的其他错误情况。  
要启用看门狗功能，请使用 DEV\_SM\_WATCHDOG。

### 12.7.3. 架构

#### 12.7.3.1 时钟速度

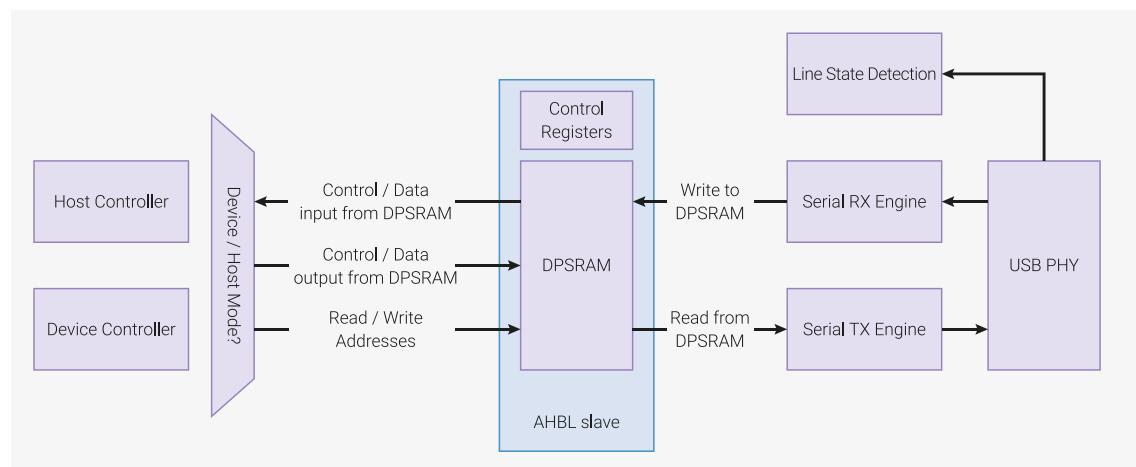
该控制器要求 `clk_usb` 以 48MHz 运行。

**注意**

`clk_sys` 也必须以超过 48MHz 的频率运行，详见 RP2350-E12。

#### 12.7.3.2 概述

图124 USB  
控制器架构的简化  
概述。



USB控制器采用面积高效设计，将设备控制器或主机控制器复用到一组公共组件上。各组件详述如下。

### 12.7.3.3 USB PHY

USB PHY提供USB DP和DM引脚与控制器数字逻辑之间的电气接口。DP和DM引脚为差分对，意味着其信号值始终相反，除非用于编码特定线路状态（例如SE0）。USB PHY驱动DP和DM引脚传输数据，并对接收的数据进行差分接收。USB PHY向线路状态检测模块提供单端和差分接收的数据。

USB PHY内置上拉和下拉电阻。当控制器作为全速设备时，DP引脚被上拉，以向主机指示已连接全速设备。在主机模式下，对DP和DM引脚施加弱下拉，使线路保持逻辑低电平，直到设备对DP（全速）或DM（低速）进行上拉。

### 12.7.3.4. 线路状态检测

USB 2.0规范定义了多个线路状态（总线复位、已连接、挂起、恢复、数据1、数据0等），需被检测。线路状态检测模块包含多个状态机，用于检测这些状态并向其他硬件组件发送事件信号。USB中无共享时钟信号，接收数据必须由内部时钟采样。USB全速最大数据速率为12 Mb/s。接收数据以48 MHz采样，提供4个时钟周期用于采集和滤除总线状态。线路状态检测模块将已滤波的RX数据分配至串行RX引擎。

### 12.7.3.5. 串行RX引擎

串行接收（RX）引擎解码线路状态检测模块捕获的接收数据。其产生以下信息：

- 传入数据包的PID
- 传入数据的设备地址
- 传入数据的设备端点
- 数据字节

串行接收引擎还通过对传入数据执行CRC校验以检测RX数据中的错误。任何错误均会向其他硬件模块发出信号，并可能触发中断。

### ⓘ 注意

若在主机或设备模式下传输数据包过程中断开USB电缆，硬件将报告错误。  
若启用错误中断，软件必须考虑此情况。

#### 12.7.3.6. 串行TX引擎

串行发送（TX）引擎是串行接收引擎的镜像。其连接至当前活动控制器（设备或主机）。其创建 TOKEN和 DATA包，计算CRC，并在总线上发送。

#### 12.7.3.7. DPSRAM

USB控制器使用4KB（4096字节）的双端口SRAM（DPSRAM）用于存储控制寄存器和数据缓冲区。DPSRAM可作为USB控制器地址0（[0x50100000](#)）处的32位宽存储器访问。

DPSRAM具备以下特性，这些特性与RP2350上的大多数寄存器有所不同：

- 支持8位、16位及32位访问（一般而言，RP2350寄存器仅支持32位访问）
- 不支持置位/清除别名。（通常，RP2350寄存器支持此类功能）

数据缓冲区通常为64字节长度，此为大多数全速数据包的最大正常包大小。

等时端点支持最大1023字节缓冲区大小。其他数据包类型的每个缓冲区最大为64字节。

##### 12.7.3.7.1. 并发访问

USB控制器中的DPSRAM为异步运行。名称中的**dual port**表明处理器和USB控制器均配备读写端口，并且这两个端口处于不同的时钟域。因此，处理器和USB控制器可同时访问相同的内存地址。一方可同时写入，而另一方可同时读取。这可能导致数据读取不一致。您可通过遵循本节所述规则避免此类情况发生。

缓冲区控制寄存器中的 **AVAILABLE**位指示缓冲区的所有权归属。处理器将此位设置为1，以将缓冲区所有权赋予控制器。控制器使用完缓冲区后，会将该位复位为0。应将 **AVAILABLE**位与缓冲区控制寄存器中的其他数据分开放置，确保设置 **AVAILABLE**位时其他数据依然准确。

这是因为处理器时钟 **clk\_sys**的频率可能是 **clk\_usb**时钟的数倍。因此，**clk\_sys**能够在较低频率时钟的USB控制器读取期间更新数据。正确的流程如下：

- 将缓冲区信息（长度等）写入缓冲区控制寄存器。
- 执行 **nop**指令，延续若干个 **clk\_sys**周期，以确保至少经历一个 **clk\_usb**周期。考虑一种情况，**clk\_sys**运行于125MHz，**clk\_usb**运行于48MHz。由于  $\lceil \frac{125}{48} \rceil = 3$ ，你应在两次写入之间发出3条 **nop**指令，以确保至少经历一个 **clk\_usb**周期。
- 设置 **AVAILABLE**位。

若 **clk\_sys**与 **clk\_usb**频率相同，则无需单独设置 **AVAILABLE**位。

### ⓘ 注意

当 USB 控制器将状态写回 DPSRAM 时，对缓冲区 0 执行低 2 字节的 16 位写操作，对缓冲区 1 执行高 2 字节的 16 位写操作。使用双缓冲模式时，软件更新缓冲区控制寄存器应始终将其视作两个独立的 16 位寄存器。

#### 12.7.3.7.2. 布局

地址 `0x0 → 0xff` 用于存放配置信息的控制寄存器。剩余空间，地址 `0x100 → 0xffff` (3840字节) 可用于数据缓冲区。控制器的控制寄存器起始于地址

`0x10000`。

内存布局取决于USB控制器的工作模式：

- 在设备模式下，主机会访问多个端点，因此每个端点必须拥有端点控制寄存器和缓冲区控制寄存器。
- 在主机模式下，运行于处理器上的主机软件决定访问哪些端点和设备。这仅需要一套端点控制寄存器和缓冲区控制寄存器。除了软件驱动的数据传输外，主机控制器可以轮询最多15个中断端点，并为每个中断端点配置相应寄存器。

表1192. DPSRAM  
布局

偏移量	设备功能	主机功能
<code>0x0</code>	设置包 (8字节)	
<code>0x8</code>	EP1 入端点控制	中断端点控制寄存器1
<code>0xc</code>	EP1 输出控制	备用
<code>0x10</code>	EP2 输入控制	中断端点控制2
<code>0x14</code>	EP2 输出控制	备用
<code>0x18</code>	EP3 输入控制	中断端点控制3
<code>0x1c</code>	EP3 输出控制	备用
<code>0x20</code>	EP4 输入控制	中断端点控制4
<code>0x24</code>	EP4 输出控制	备用
<code>0x28</code>	EP5 输入控制	中断端点控制5
<code>0x2c</code>	EP5 输出控制	备用
<code>0x30</code>	EP6 输入控制	中断端点控制6
<code>0x34</code>	EP6 输出控制	备用
<code>0x38</code>	EP7 输入控制	中断端点控制7
<code>0x3c</code>	EP7 输出控制	备用
<code>0x40</code>	EP8 输入控制	中断端点控制8
<code>0x44</code>	EP8 输出控制	备用
<code>0x48</code>	EP9 输入控制	中断端点控制9
<code>0x4c</code>	EP9 输出控制	备用
<code>0x50</code>	EP10 输入控制	中断端点控制10
<code>0x54</code>	EP10 输出控制	备用
<code>0x58</code>	EP11 输入控制	中断端点控制11

偏移量	设备功能	主机功能
0x5c	EP11 输出控制	备用
0x60	EP12 输入控制	中断端点控制12
0x64	EP12 输出控制	备用
0x68	EP13 输入控制	中断端点控制 13
0x6c	EP13 输出控制	备用
0x70	EP14 输入控制	中断端点控制 14
0x74	EP14 输出控制	备用
0x78	EP15 输入控制	中断端点控制 15
0x7c	EP15 输出控制	备用
0x80	EP0 输入缓冲控制	EPx 缓冲控制
0x84	EP0 输出缓冲控制	备用
0x88	EP1 输入缓冲控制	中断端点缓冲控制 1
0x8c	EP1 输出缓冲控制	备用
0x90	EP2 输入缓冲控制	中断端点缓冲控制 2
0x94	EP2 输出缓冲控制	备用
0x98	EP3 输入缓冲控制	中断端点缓冲区控制3
0x9c	EP3 出端缓冲区控制	备用
0xa0	EP4 入端缓冲区控制	中断端点缓冲区控制4
0xa4	EP4 出端缓冲区控制	备用
0xa8	EP5 入端缓冲区控制	中断端点缓冲区控制5
0xac	EP5 出端缓冲区控制	备用
0xb0	EP6 入端缓冲区控制	中断端点缓冲区控制6
0xb4	EP6 出端缓冲区控制	备用
0xb8	EP7 入端缓冲区控制	中断端点缓冲区控制7
0xbc	EP7 出端缓冲区控制	备用
0xc0	EP8 入端缓冲区控制	中断端点缓冲区控制8
0xc4	EP8 出端缓冲区控制	备用
0xc8	EP9 入端缓冲区控制	中断端点缓冲区控制9
0xcc	EP9 出站缓冲区控制	备用
0xd0	EP10 入站缓冲区控制	中断端点缓冲区控制10
0xd4	EP10 出站缓冲区控制	备用
0xd8	EP11 入站缓冲区控制	中断端点缓冲区控制11
0xdc	EP11 出站缓冲区控制	备用
0xe0	EP12 入站缓冲区控制	中断端点缓冲区控制12
0xe4	EP12 出站缓冲区控制	备用
0xe8	EP13 入站缓冲区控制	中断端点缓冲区控制13

偏移量	设备功能	主机功能
0xec	EP13 出站缓冲区控制	备用
0xf0	EP14 入站缓冲区控制	中断端点缓冲区控制14
0xf4	EP14 出站缓冲区控制	备用
0xf8	EP15 入站缓冲区控制	中断端点缓冲区控制 15
0xfc	EP15 输出缓冲区控制	备用
0x100	EP0 缓冲区 0 (输入和输出共享)	EPx 控制
0x140	可选 EP0 缓冲区 1	备用
0x180	数据缓冲区	

#### 12.7.3.7.3. 端点控制寄存器

端点控制寄存器用于配置端点，包括：

- 端点类型
- 端点数据缓冲区的基址（双缓冲时为数据缓冲区）
- 触发控制器中断输出的端点事件

设备必须支持端点0，以便响应 **SETUP**包并完成枚举。因此，端点0不存在端点控制寄存器。其缓冲区起始地址为 0x100。所有其他端点可配置为单缓冲或双缓冲，并映射至所编程的基址。由于EP0无端点控制寄存器，其中断使能控制由 SIE\_CTRL 提供。

表 1193。端点控制寄存器布局

Bit (位)	设备功能	主机功能
31	端点启用	
30	单缓冲 (64字节) = 0, 双缓冲 (64字节×2) = 1	
29	为每个传输缓冲区启用中断	
28	为每两个传输缓冲区启用中断 (仅适用于双缓冲)	
27:26	端点类型：控制 = 0, 等时 = 1, 批量 = 2, 中断 = 3	
25:18	不适用	主机控制器轮询该端点的间隔时间。仅适用于中断端点。单位为毫秒，值减1。例如：值为9表示每10毫秒轮询一次该端点。
17	中断触发于 <b>STALL</b>	
16	中断触发于 <b>NAK</b>	
15:6	数据缓冲区在DPSRAM中的基址偏移量	

##### 注意

数据缓冲区基地址必须64字节对齐，因位0至5被忽略。

#### 12.7.3.7.4. 缓冲区控制寄存器

缓冲区控制寄存器包含该端点数据缓冲区的状态信息。该寄存器由处理器与控制器共享。如果端点配置为单缓冲，则仅使用缓冲区的前半部分（第0至15位）。

若为双缓冲，则缓冲区选择从缓冲区0开始。此后，缓冲区选择在缓冲区0和缓冲区1之间切换，

除非设置了复位缓冲区选择位（该位将缓冲区选择重置为缓冲区0）。缓冲区选择的值为控制器内部状态，处理器不可访问。

对于主机中断和等时包于 EPx上的传输，即使传输失败，缓冲区满位在完成时仍被置位。要确定错误，请读取SIE\_STATU S寄存器中的错误位。

表1194。缓冲区控制寄存器布局

Bit (位)	功能
31	缓冲区1已满。对于 IN事务，处理器应将其置1；对于 OUT事务，应置0。控制器在 OUT事务中将其置为1，表明缓冲区已填满。 控制器将其设置为0，用于 IN事务，因为缓冲区已清空。仅在双缓冲模式下有效。
30	缓冲区1的最后一个传输缓冲。仅在双缓冲模式下有效。
29	缓冲区1的数据PID - DATA0 = 0, DATA1 = 1。仅在双缓冲模式下有效。
27:28	同步传输模式下的双缓冲偏移量 (0 = 128, 1 = 256, 2 = 512, 3 = 1024)。
26	缓冲区1可用。指示该缓冲区是否可被控制器用于传输。处理器在配置缓冲区时将其设置为1。控制器在将数据发送至主机以进行 IN事务，或从主机接收数据填充缓冲区以进行OUT事务后，将其设置为0。仅在双缓冲模式下有效。
25:16	缓冲区1传输长度。仅在双缓冲模式下有效。
15	缓冲区0已满。对于 IN事务，处理器应将其置1；对于 OUT事务，应置0。控制器在 OUT事务中将其置为1，表明缓冲区已填满。 控制器将其设置为0，用于 IN事务，因为缓冲区已清空。
14	缓冲区0的最后一个传输缓冲。
13	缓冲区0的数据PID—DATA0 = 0, DATA1 = 1。
12	将缓冲区选择重置为缓冲区0—在传输结束时清除，仅限设备使用。
11	向设备发送STALL，主机接收STALL。
10	缓冲区0可用。指示控制器是否可使用该缓冲区进行传输。 处理器在配置缓冲区时将其设置为1。控制器在向主机发送 IN事务数据后，或为 OUT事务从主机填充数据至缓冲区后，将此位清零。
9:0	缓冲区0传输长度。

### ● 警告

若您以不同速度运行 clk\_sys 和 clk\_usb，请在缓冲区控制寄存器中的其他数据之后设置available和stall位。否则，控制器可能会启动包含来自先前数据包的数据的事务。控制器可能检测到available位被设置，但获取的是先前数据包的数据PID或长度。

### 12.7.3.8. 设备控制器

本节详细说明设备控制器接收主机各类数据包时的操作方式。

#### 12.7.3.8.1. SETUP

设备控制器必须始终接受来自主机的SETUP数据包。DPSRAM将其前8个字节专用于存放设置数据包。

USB 2.0规范规定，接收设置数据包同时清除EP0上的任何阻塞位。因此，阻塞

位于EP0端点通过EP\_STALL\_ARM寄存器中的两个位进行控制。当接收到设置数据包时，这些位将被清除。这意味着，要在EP0端点发送阻塞信号，必须同时设置缓冲控制寄存器中的阻塞位和EP\_STALL\_ARM寄存器中的相应位。

在无误的情况下，设置数据包将被存放在DPSRAM偏移量0x0处的设置数据包缓冲区。设备控制器随后将回复ACK信号。

最后，SIE\_STATUS.SETUP\_REC位被置位，以指示已接收到设置数据包。如果程序员已启用 SETUP\_REC中断（参见INTE），则会触发中断。

#### 12.7.3.8.2. IN

从设备的角度来看，IN传输是指将数据传输到主机。当接收到来自主机的IN令牌时，请按以下方式处理：

**TOKEN** 阶段：

1. 如果缓冲区控制寄存器中设置了 STALL（且对于 EP0，相应的EP\_STALL\_ARM位已设置），则发送STALL响应并进入空闲状态。
2. 如果缓冲区控制中的 AVAILABLE 和 FULL 位均被设置，则进入 DATA阶段。
3. 若为等时端点，则进入空闲状态。
  - 否则，发送 NAK并进入 DATA阶段。

**DATA** 阶段：

1. 发送数据。
2. 若为等时端点，则进入空闲状态。
  - 否则，进入 ACK阶段。

**ACK** 阶段：

1. 等待来自主机的 ACK数据包。
2. 若发生超时，则触发超时错误。
3. 如果收到 ACK，数据包传输完成，则进入 STATUS阶段。

**STATUS** 阶段：

1. 若该缓冲区为传输中的最后一个（即缓冲区控制寄存器中的LAST\_BUFFER位已设置），则置位 SIE\_STATUS.TRANSACTION\_COMPLETE。
2. 若端点采用双缓冲，则切换缓冲区选择至另一缓冲区。
3. 在 BUFF\_STATUS 中设置相应位以指示缓冲区已完成。处理此事件时，程序应读取 BUFF\_CPU\_SHOULD\_HANDLE 以确定完成的是缓冲区 0 还是缓冲区 1。若端点为双缓冲，则两个缓冲区均可能已完成。被清除的 BUFF\_STATUS 位会被重新置位，且 BUFF\_CPU\_SHOULD\_HANDLE 会相应变化。
4. 在缓冲区控制寄存器的相应半区更新状态：length、pid 与 last\_buff 被设置，其余全部写入零。

若主机收到 NAK，将稍后重试。

#### 12.7.3.8.3. OUT

当从主机接收到 OUT令牌时，请按以下方式处理：

**TOKEN** 阶段：

1. 如果这不是一个等时端点，且数据 PID 与缓冲区控制寄存器不匹配，则触发

**SIE\_STATUS.DATA\_SEQ\_ERROR** (等时数据始终以 DATA0 PID 发送)。

2. 如果 **AVAILABLE**位被置位且 **FULL**位被清除，则进入 **DATA**阶段；除非 **STALL**位被置位，此时设备控制器将回复一个 **STALL**。

**DATA** 阶段：

1. 将接收的数据存储于缓冲区。若为等时端点，则进入 **STATUS**阶段。否则，进入 **ACK**阶段。

**ACK** 阶段：

1. 发送 **ACK**。进入 **STATUS**阶段。

**STATUS** 阶段：

参见 **IN STATUS**阶段：[usb-device-in-status-phase]。有一点不同之处：缓冲区控制寄存器中的 **FULL**位被置位，以指示数据已接收。在 **IN**阶段，**FULL**位被清除以指示数据已发送。

#### 12.7.3.8.4. 挂起与恢复

USB 设备控制器支持挂起、恢复以及设备发起的远程恢复（由 **SIE\_CTRL.RESUME** 触发）。**SIE\_STATUS** 中含有中断/状态位。无需启用挂起和恢复中断，因为挂起与恢复对大多数设备而言无关紧要。

当设备未检测到主机发送的任何帧起始包（每隔1毫秒发送一次）时，设备将进入挂起状态。

##### ① 注意

启用挂起中断时，设备首次连接可能会产生挂起中断，但总线处于空闲状态。主机开始发送帧起始包前，总线可能空闲数毫秒。若未连接 VBUS 检测电路，设备断开时亦会出现挂起中断。

未连接 VBUS 检测电路时，无法区分断开连接与挂起状态。

#### 12.7.3.9. 主机控制器

主机控制器的设计与设备控制器类似。主机发起所有事务，因此主机始终处理其发起的事务。因此，仅存在一组端点控制和端点缓冲控制寄存器。主机控制器还配备了额外硬件，在无软件控制事务进行时，可在后台轮询中断端点。

主机需每1毫秒向设备发送保持活动包，以防止设备进入挂起状态。全速模式使用 **SOF**（帧开始）包。低速模式则使用 **EOP**（包结束）包。通过设置 **SIE\_CTRL.KEEP\_ALIVE\_EN** 和 **SIE\_CTRL.SOF\_EN** 来启用这些包。

**SIE\_CTRL** 寄存器中的若干位用于启动主机事务：

- **SEND\_SETUP**- 发送设置包。通常与**RECEIVE\_TRANS**配合使用，先发送设置包，随后执行设备预期的额外数据事务。
- **SEND\_TRANS** - 此传输为 **OUT** 方向，来源于主机。
- **RECEIVE\_TRANS** - 此传输为 **IN** 方向，进入主机。
- **START\_TRANS** - 启动传输（非锁存）。
- **STOP\_TRANS** - 停止当前传输（非锁存）。
- **PREAMBLE\_ENABLE**- 用于向全速集线器上的低速设备发送数据包。主机会在发送的每个数据包前附加一个 **PRE**令牌包（即 **PRE**、**TOKEN**、**PRE**、**DATA**、**pre**、**ACK**）。
- **SOF\_SYNC**- 用于延迟事务，直到下一个 **SOF**之后。适用于中断端点和同步端点。主机控制器防止64字节长度的事务与 **SOF**数据包冲突。对于更长的同步

数据包，软件层负责避免冲突。为避免软件冲突，请使用 **SOF\_SYNC** 并限制每帧发送的数据包数量。如果事务由多个数据包组成，**SOF\_SYNC** 仅适用于第一个数据包。

**START\_TRANS** 位与 **SIE\_CTRL** 寄存器中的其他控制位分别同步，因为处理器时钟 **clk\_sys** 可能与 **clk\_usb** 时钟异步。务必在写入 **START\_TRANS** 和 **SIE\_CTRL** 中其他位之间至少经过两个 **clk\_usb** 周期。此举确保当控制器被触发启动传输时，寄存器内容保持稳定。

考虑一种情况，**clk\_sys** 运行于 125MHz，**clk\_usb** 运行于 48MHz。由于  $\lceil \frac{125}{48} \rceil \times 2 = 6$  应在写入之间插入 6 条 **nop** 指令，以确保至少经过两个 **clk\_usb** 周期。

#### 12.7.3.9.1. SETUP

主机发送的 **SETUP** 数据包始终来源于 DPSRAM 偏移地址 **0x0** 处专用的 8 字节空间。

与设备控制器类似，**setup** 数据包不对应任何控制寄存器。当写入 **START\_TRANS** 且设置了 **SEND\_SETUP** 位时，参数将被硬编码并加载至硬件。一旦设置包发送完毕，主机状态机将等待设备的 **ACK** 响应。若发生超时，将触发一个 **RX\_TIMEOUT** 错误。

若设置了 **SEND\_TRANS** 位，主机状态机将进入 **OUT** 阶段。通常，**SEND\_SETUP** 包与 **RECEIVE\_TRANS** 位共同使用，因此控制器在发送设置包后进入 **IN** 阶段。

#### 12.7.3.9.2. IN

当设置 **START\_TRANS** 且 **RECEIVE\_TRANS** 位时，将触发一次 **IN** 传输。如设置了 **SEND\_SETUP** 位，可能会先发送一个 **SETUP** 包。

**CONTROL** 阶段：

1. 读取位于 **0x80** 地址的 **EPx** 控制寄存器以获取以下端点信息：
  - 是否为双缓冲？
  - 启用了哪些中断？
  - 数据缓冲区的基址（双缓冲模式下为数据缓冲区）
  - 端点类型是什么？
2. 读取位于 **0x100** 的 **EPx** 缓冲区控制寄存器，以获取端点缓冲信息，如传输长度和数据 PID。
3. 设置 **AVAILABLE** 位（主机状态机将检查该位）。
4. 清除 **FULL** 位。

**TOKEN** 阶段：

1. 向设备发送 **IN** 令牌包。目标设备地址和端点信息来源于 **ADDR\_ENDP** 寄存器。

**DATA** 阶段：

1. 接收来自设备的第一个数据包。
2. 如果设备未响应，则触发 RX 超时错误。
3. 如果不是同步端点，且数据 PID 与缓冲区控制寄存器不匹配，则触发 **SIE\_STATUS.DATA\_SEQ\_ERROR**（同步数据始终使用 DATA0 pid 发送）。

**ACK** 阶段：

1. 向设备发送 **ACK**。

**STATUS** 阶段：

1. 设置BUFF\_STATUS位，并更新缓冲区控制寄存器。
2. 如适用，设置 FULL、DATA\_PID、WR\_LEN和 LAST\_BUFF。
3. 若这是传输中的最后一个缓冲区，请设置TRANS\_COMPLETE。

**CONTROL** 阶段（续）：

主机状态机执行 IN 事务，直到在buffer\_control 寄存器中检测到 LAST\_BUFF 标志。

若主机处于双缓冲模式，主机控制器将在缓冲区控制寄存器的 BUF0 和 BUF1部分之间切换。

否则，控制器读取缓冲区0的缓冲区控制寄存器，然后等待 FULL 清除且 AVAILABLE被置位后，开始下一次 IN 事务，在 CONTROL 阶段等待。

若主机收到零长度数据包，表示设备无更多数据。无论 LAST\_BUFF 标志是否被设置，主机状态机会停止监听更多数据。若需从主机软件检测此情况，请检查缓冲区控制寄存器中 BUFF\_DONE 对应的数据长度是否为0。

#### 12.7.3.9.3. OUT

当 START\_TRANS 被设置时，SEND\_TRANS位触发一次OUT传输。若 SEND\_SETUP位被设置，则可能先发送一个 SETUP数据包。

**CONTROL** 阶段：

1. 读取 EPx 控制寄存器以获取端点信息（同第12.7.3.9.2节）。
2. 读取 EPx 缓冲区控制寄存器以获取传输长度和数据PID。AVAILABLE 和 FULL 必须在传输开始前设置。

**TOKEN** 阶段

1. 向设备发送 OUT 包。目标设备地址和端点信息来源于ADDR\_ENDP 寄存器。

**DATA** 阶段：

1. 向设备发送第一个数据包。如果端点类型为同步传输（isochronous），则无 ACK 阶段，主控器直接进入状态阶段。若收到 ACK，则进入状态阶段。否则：
  - 若主机未收到回复，触发 SIE\_STATUS.RX\_TIMEOUT。
  - 若主机收到 NAK，触发 SIE\_STATUS.NAK\_REC，并重新发送数据包。
  - 若主机收到 STALL，触发 SIE\_STATUS.STALL\_REC，并进入空闲状态。

**STATUS** 阶段：

1. 设置 BUFF\_STATUS 位并更新缓冲区控制寄存器。FULL 将被置为0。若这是传输中的最后一个缓冲区，则将设置TRANS\_COMPLETE 标志。

**CONTROL** 阶段（续）：

1. 如非传输中的最后缓冲区，请等待 FULL 和 AVAILABLE 标志在 EPx 缓冲区控制寄存器中再次被置位。

#### 12.7.3.9.4. 中断端点

主机控制器最多可轮询15个中断端点。启用中断端点，程序员必须：

- 在主机控制器上选择下一个空闲的中断端点槽（起始编号为1，最大可达15）。
- 如同处理普通的 IN 或 OUT 传输，设置相应的端点控制寄存器和缓冲区控制寄存器。由于中断端点为单缓冲，缓冲区控制寄存器中的BUF1位无效。
- 在对应的 ADDR\_ENDP 寄存器（ADDR\_ENDP1 至 ADDR\_ENDP15）中设置设备地址及端点号。

如果设备为低速且连接至全速集线器，应设置前导码位，同时设置端点方向位。

- 在INT\_EP\_CTRL中设置相应中断端点的活动位（位1至位15之一）。

通常，中断端点使用 IN传输。主机可能轮询USB集线器，以检测其任一端口状态是否发生变化。若无变化，集线器向控制器返回 NAK，且不会执行任何操作。同样，除非自上次轮询该中断端点以来鼠标已移动，否则鼠标会回复 NAK。

主机控制器发送 SOF数据包后，控制器将对中断端点进行轮询。

控制器从1循环至15，尝试轮询INT\_EP\_CTRL中将 EP\_ACTIVE位设置为1的任何中断端点。随后，控制器读取端点控制寄存器与缓冲区控制寄存器，以确认是否存在可用缓冲区（即，对于OUT传输，缓冲区应为FULL且AVAILABLE；对于IN传输，应为NOT FULL且AVAILABLE）。若无，则控制器将切换至下一个中断端点槽。

如有可用缓冲区，传输将按常规 IN或 OUT传输处理，当中断端点拥有有效缓冲区时，BUFF\_STATUS中的 BUFF\_DONE标志将被置位。

### 12.7.3.10. VBUS 控制

USB 控制器可连接至 GPIO 引脚（详见第9章）以实现以下 VBUS 控制：

- **VBUS 使能**，用于主机模式下启用 VBUS。在 SIE\_CTRL 中设置。
- **VBUS 检测**，用于设备模式下检测 VBUS 是否存在。通过 SIE\_STATUS 中的某个位设置。亦可触发 INTE 中启用的 VBUS\_DETECT中断。
- **VBUS 过流**，用于检测过流事件。适用于设备模式和主机模式。VBUS 过流为 SIE\_STATUS 中的一个位。

无须将这些引脚连接至 GPIO。主机可持续供应 VBUS，且当 DP或 DM引脚被拉高时，能检测到设备连接。VBUS 检测可通过 USB\_PWR 强制执行。

### 12.7.4. 程序员模型

#### 12.7.4.1. TinyUSB

RP2350 TinyUSB 端口为该 USB 控制器的参考实现。该端口可在 [pico-sdk GitHub](#) 仓库的以下文件中找到：

```
dcd_rp2040.c
hcd_rp2040.c
rp2040_usb.h
```

#### 12.7.4.2. 独立设备示例

独立 USB 设备示例dev\_lowlevel便于理解如何与 USB 控制器交互，无需掌握 TinyUSB 的抽象层。除端点 0 外，该独立设备还有两个批量端点：EP1 OUT和EP2 IN。此设备设计为将其在 EP1接收的任何数据发送到 EP2。该示例附带一个小型 Python 脚本，向 EP1写入“Hello World”，并验证其是否正确接收至 EP2。

本节中包含的代码解释了如何设置USB设备控制器以实现接收功能。同时展示了软件如何响应从主机接收到的设置包。

图125。 dev\_lowlevel USB设备示例的USB分析仪跟踪。控制传输即设备枚举过程。第一个批量OUT（主机输出）传输，用蓝色标记，表示主机向设备发送“Hello World”。第二个批量/N（主机输入）传输，表示设备向主机返回“Hello World”。

00	Packet	H	Reset	15.006 ms
01	Transfer	F	Control	ADDR ENDP
0	S	GET	0	0
02	Packet	H	Reset	15.006 ms
03	Transfer	F	Control	ADDR ENDP
1	S	SET	0	0
04	Transfer	F	Control	ADDR ENDP
2	S	GET	7	0
05	Transfer	F	Control	ADDR ENDP
3	S	GET	7	0
06	Transfer	F	Control	ADDR ENDP
4	S	GET	7	0
07	Transfer	F	Control	ADDR ENDP
5	S	GET	7	0
08	Transfer	F	Control	ADDR ENDP
6	S	GET	7	0
09	Transfer	F	Control	ADDR ENDP
7	S	GET	7	0
10	Transfer	F	Bulk	ADDR ENDP
8	S	OUT	7	1
11	Transfer	F	Bulk	ADDR ENDP
9	S	IN	7	2

#### 12.7.4.2.1. 设备控制器初始化

以下代码用于初始化USB设备：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\\_lowlevel/dev\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c) 第183至217行

```

183 void usb_device_init() {
184 // 重置USB控制器
185 reset_unreset_block_num_wait_blocking(RESET_USBCTRL);
186
187 // 清除dpram中任何先前状态以防意外
188 memset(usb_dpram, 0, sizeof(*usb_dpram)); ①
189
190 // 在处理器上启用 USB 中断
191 irq_set_enabled(USBCTRL_IRQ, true);
192
193 // 将控制器复用至板载 USB 物理层
194 usb_hw->muxing = USB_USB_MUXING_TO_PHY_BITS | USB_USB_MUXING_SOFTCON_BITS;
195
196 // 强制VBUS 检测，使设备误认为已连接至主机
197 usb_hw->pwr = USB_USB_PWR_VBUS_DETECT_BITS | USB_USB_PWR_VBUS_DETECT_OVERRIDE_EN_BITS;
198
199 // 以设备模式启用 USB 控制器
200 usb_hw->main_ctrl = USB_MAIN_CTRL_CONTROLLER_EN_BITS;
201
202 // 为每次 EP0 事务启用中断
203 usb_hw->sie_ctrl = USB_SIE_CTRL_EP0_INT_1BUF_BITS; ②
204
205 // 启用中断，适用于缓冲区完成、总线重置以及接收设置包时
206 //
207 usb_hw->inte = USB_INTS_BUFF_STATUS_BITS |
208 USB_INTS_BUS_RESET_BITS |
209 USB_INTS_SETUP_REQ_BITS;
210
211 // 配置端点（端点控制寄存器）
212 // 由设备配置描述
213 usb_setup_endpoints();
214
215 // 通过在DP线上使能上拉电阻展示全速设备

```

```
216 usb_hw_set->sie_ctrl = USB_SIE_CTRL_PULLUP_EN_BITS;
217 }
```

#### 12.7.4.2.2. 配置 EP1和 EP2的端点控制寄存器

函数`usb_configure_endpoints`遍历设备配置中定义的所有端点（包括没有端点控制寄存器的 EP0输入和 EP0输出）并调用`usb_configure_endpoint`函数。该操作用于配置该端点的端点控制寄存器：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\\_lowlevel/dev\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c) 第149-164行

```
149 void usb_setup_endpoint(const struct usb_endpoint_configuration *ep) {
150 printf("设置端点 0x%x, 缓冲区地址 0x%p\n", ep->descriptor->bEndpointAddress, ep->data_buffer); 151
152
153 // EP0 无缓冲区, 若如此则返回
154 if (!ep->endpoint_control) {
155 return;
156 }
157
158 // 获取数据缓冲区在 USB 控制器 DPRAM 中的偏移量
159 uint32_t dpram_offset = usb_buffer_offset(ep->data_buffer);
160 uint32_t reg = EP_CTRL_ENABLE_BITS
161 | EP_CTRL_INTERRUPT_PER_BUFFER
162 | (ep->descriptor->bmAttributes << EP_CTRL_BUFFER_TYPE_LSB)
163 | dpram_offset;
164 *ep->endpoint_control = reg;
165 }
```

#### 12.7.4.2.3. 接收设置包

当接收到设置包时，会触发中断，故中断处理程序必须对此事件进行处理：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\\_lowlevel/dev\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c) 第494至504行

```
494 void isr_usbcctrl(void) {
495 // USB中断处理函数
496 uint32_t status = usb_hw->ints;
497 uint32_t handled = 0;
498
499 // 已接收设置包
500 if (status & USB_INTS_SETUP_REQ_BITS) {
501 handled |= USB_INTS_SETUP_REQ_BITS;
502 usb_hw_clear->sie_status = USB_SIE_STATUS_SETUP_REC_BITS;
503 usb_handle_setup_packet();
504 }
505 }
```

控制器将`SETUP`包写入DPSRAM的前8字节，因此设置包处理程序将该内存区域转换为`struct usb_setup_packet *`类型：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\\_lowlevel/dev\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c) 第383至427行

```
383 void usb_handle_setup_packet(void) {
384 volatile struct usb_setup_packet *pkt = (volatile struct usb_setup_packet *) &usb_dpram
385 ->setup_packet;
386 uint8_t req_direction = pkt->bmRequestType;
```

```

386 uint8_t req = pkt->bRequest;
387
388 // 将EP0 IN的PID重置为1
389 usb_get_endpoint_configuration(EP0_IN_ADDR)->next_pid = 1u;
390
391 if (req_direction == USB_DIR_OUT) {
392 if (req == USB_REQUEST_SET_ADDRESS) {
393 usb_set_device_address(pkt);
394 } else if (req == USB_REQUEST_SET_CONFIGURATION) {
395 usb_set_device_configuration(pkt);
396 } else {
397 usb_acknowledge_out_request();
398 printf("其他 OUT 请求 (0x%x)\r\n", pkt->bRequest);
399 }
400 } else if (req_direction == USB_DIR_IN) {
401 if (req == USB_REQUEST_GET_DESCRIPTOR) {
402 uint16_t descriptor_type = pkt->wValue >> 8;
403
404 switch (descriptor_type) {
405 case USB_DT_DEVICE:
406 usb_handle_device_descriptor(pkt);
407 printf("获取设备描述符\r\n");
408 break;
409
410 case USB_DT_CONFIG:
411 usb_handle_config_descriptor(pkt);
412 printf("获取配置描述符\r\n");
413 break;
414
415 case USB_DT_STRING:
416 usb_handle_string_descriptor(pkt);
417 printf("获取字符串描述符\r\n");
418 break;
419
420 default:
421 printf("未处理的 GET_DESCRIPTOR 类型 0x%x\r\n", descriptor_type);
422 }
423 } else {
424 printf("其他 IN 请求 (0x%x)\r\n", pkt->bRequest);
425 }
426 }
427 }

```

#### 12.7.4.2.4. 回复 EP0 IN端点上的设置包

主机首先请求设备描述符。以下代码处理该设置请求：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\\_lowlevel/dev\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c), 第266至273行

```

266 void usb_handle_device_descriptor(volatile struct usb_setup_packet *pkt) {
267 const struct usb_device_descriptor *d = dev_config.device_descriptor;
268 // EP0 IN端点
269 struct usb_endpoint_configuration *ep = usb_get_endpoint_configuration(EP0_IN_ADDR);
270 // 始终响应pid为1
271 ep->next_pid = 1;
272 usb_start_transfer(ep, (uint8_t *) d, MIN(sizeof(struct usb_device_descriptor), pkt-
273 >wLength));
273 }

```

usb\_start\_transfer函数将待发送数据复制至相应硬件缓冲区，并对缓冲区进行配置。

控制寄存器。缓冲区控制寄存器一旦被写入，设备控制器即向主机响应数据。在此之前，设备将回复 NAK：

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\\_lowlevel/dev\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c) 第238至260行

```

238 void usb_start_transfer(struct usb_endpoint_configuration *ep, uint8_t *buf, uint16_t len) {
239 // 为简化示例，断言长度不超过64字节。
240 // 多包传输请参见 tinyusb 移植层。
241 assert(len <= 64);
242
243 printf("开始传输，长度为 %d，端点地址为 0x%x\n", len, ep->descriptor->bEndpointAddress); 244
244
245 // 准备缓冲区控制寄存器的值
246 uint32_t val = len | USB_BUF_CTRL_AVAIL;
247
248 if (ep_is_tx(ep)) {
249 // 需要将数据从用户缓冲区复制到USB存储器
250 memcpy((void *) ep->data_buffer, (void *) buf, len);
251 // 标记为已满
252 val |= USB_BUF_CTRL_FULL;
253 }
254
255 // 设置pid并切换以用于下一次传输
256 val |= ep->next_pid ? USB_BUF_CTRL_DATA1_PID : USB_BUF_CTRL_DATA0_PID;
257 ep->next_pid ^= 1u;
258
259 *ep->buffer_control = val;
260 }
```

## 12.7.5. 寄存器列表

USB寄存器起始地址为 **0x50110000**（在SDK中定义为USBCTRL\_REGS\_BASE）。

表1195。  
USB寄存器列表

偏移量	名称	说明
0x000	ADDR_ENDP	设备地址及端点控制
0x004	ADDR_ENDP1	中断端点1。仅适用于主机模式。
0x008	ADDR_ENDP2	中断端点2。仅适用于主机模式。
0x00c	ADDR_ENDP3	中断端点3。仅适用于主机模式。
0x010	ADDR_ENDP4	中断端点4。仅适用于主机模式。
0x014	ADDR_ENDP5	中断端点5。仅适用于主机模式。
0x018	ADDR_ENDP6	中断端点6。仅适用于主机模式。
0x01c	ADDR_ENDP7	中断端点7。仅适用于主机模式。
0x020	ADDR_ENDP8	中断端点8。仅适用于主机模式。
0x024	ADDR_ENDP9	中断端点9。仅适用于主机模式。
0x028	ADDR_ENDP10	中断端点10。仅适用于主机模式。
0x02c	ADDR_ENDP11	中断端点11。仅适用于主机模式。
0x030	ADDR_ENDP12	中断端点12。仅适用于主机模式。
0x034	ADDR_ENDP13	中断端点13。仅适用于主机模式。

偏移量	名称	说明
0x038	ADDR_ENDP14	中断端点14。仅适用于主机模式。
0x03c	ADDR_ENDP15	中断端点15。仅适用于主机模式。
0x040	MAIN_CTRL	主控寄存器
0x044	SOF_WR	设置主机控制器中的SOF（帧开始）帧编号。 SOF包每1毫秒发送一次，主机会每次将帧编号递增1。
0x048	SOF_RD	读取最近一次检测到的SOF（帧开始）帧编号。设备模式下，接收到的主机最后一个SOF帧编号。主机模式下，主机发送的最后一个SOF帧编号。
0x04c	SIE_CTRL	SIE控制寄存器
0x050	SIE_STATUS	SIE状态寄存器
0x054	INT_EP_CTRL	中断端点控制寄存器
0x058	BUFF_STATUS	缓冲区状态寄存器。此位被设置表示该端点的缓冲区已完成（前提是已启用缓冲区中断）。可能会同时完成两个缓冲区，因此清除缓冲区状态位后，该位可能在下一个时钟周期立即重新被置位。
0x05c	BUFF_CPU_SHOULD_HANDLE	应处理哪一个双缓冲区。仅在每个缓冲区产生中断时有效（即非每两个缓冲区产生一次中断）。对主机中断端点轮询无效，因为它们仅为单缓冲。
0x060	EP_ABORT	仅设备端：可设置为忽略该端点的缓冲区控制寄存器，以便撤销缓冲区。在此位被清除之前，每次访问该端点均会发送NAK。当可安全修改缓冲区控制寄存器时，EP_ABORT_DONE寄存器中的相应位将被置位。
0x064	EP_ABORT_DONE	仅设备端：与EP_ABORT配合使用。端点空闲时设置，以告知程序员可安全修改缓冲区控制寄存器。
0x068	EP_STALL_ARM	设备：此位必须与缓冲区控制寄存器中的STALL位同时设置，以便在EP0上发送STALL信号。设备控制器在接收到SETUP包时会清除这些位，因为USB规范要求接收到SETUP包时必须清除STALL状态。
0x06c	NAK_POLL	由主机控制器使用。设置设备回复NAK时重试前的等待时间，单位为微秒。
0x070	EP_STATUS_STALL_NAK	设备：当IRQ_ON_NAK或IRQ_ON_STALL位被设置时，该位也被置位。对于EP0，该信号源自SIE_CTRL。对于所有其他端点，该信号来源于端点控制寄存器。
0x074	USB_MUXING	USB控制器的连接位置。默认应为to_phy。
0x078	USB_PWR	当VBUS信号未接入GPIO时，对电源信号进行覆盖。先设置覆盖值，再启用覆盖，切换至覆盖值。

偏移量	名称	说明
0x07c	USBPHY_DIRECT	此寄存器允许对USB物理层直接控制。需与usbphy_direct_override寄存器配合使用以启用相应覆盖位。
0x080	USBPHY_DIRECT_OVERRIDE	usbphy_direct 中各控制项的覆盖使能
0x084	USBPHY_TRIM	用于调整 USB 物理层下拉电阻的微调值。
0x088	LINESTATE_TUNING	仅供调试使用。
0x08c	中断	原始中断
0x090	INTE	中断使能
0x094	INTF	中断触发
0x098	INTS	掩码与强制后的中断状态
0x100	SOF_TIMESTAMP_RAW	仅限设备模式。48MHz 下自由运行物理层时钟计数器的原始值。用于计算 SOF 事件间时间。
0x104	SOF_TIMESTAMP_LAST	仅限设备模式。上次 SOF 事件发生时 48MHz 自由运行物理层时钟计数器的值。
0x108	SM_STATE	
0x10c	EP_TX_ERROR	每端点的发送错误计数。向各字段写入以将计数器清零。
0x110	EP_RX_ERROR	每端点的接收错误计数。向各字段写入以将计数器清零。
0x114	DEV_SM_WATCHDOG	看门狗：若设备长时间处于非空闲状态，强制状态机置于空闲并触发中断。计数器在每次状态转换时复位。 先在 enable 低电平时设置限制值，再设使能。

## USB: ADDR\_ENDP 寄存器

偏移: 0x000

### 描述

设备地址及端点控制

表 1196。  
ADDR\_ENDP 寄存器

位	描述	类型	复位
31:20	保留。	-	-
19:16	<b>ENDPOINT:</b> 设备端点，用以发送数据。仅适用于主机模式。	读写	0x0
15:7	保留。	-	-
6:0	<b>ADDRESS:</b> 设备模式下设备应响应的地址，由主机通过 SET_ADDR 设置包设定。在主机模式下，设置为要通信的设备地址。	读写	0x00

## USB: ADDR\_ENDP1, ADDR\_ENDP2, ..., ADDR\_ENDP14, ADDR\_ENDP15 寄存器

偏移量: 0x004, 0x008, ..., 0x038, 0x03c

### 描述

中断端点 N。仅适用于主机模式。

表1197。  
ADDR\_ENDP1,  
ADDR\_ENDP2, ...,  
ADDR\_ENDP14,  
ADDR\_ENDP15  
寄存器

位	描述	类型	复位
31:27	保留。	-	-
26	<b>INTEP_PREAMBLE</b> : 中断端点需要前导码 (为全速集线器上的低速设备)	读写	0x0
25	<b>INTEP_DIR</b> : 中断端点的方向。In=0, Out=1	读写	0x0
24:20	保留。	-	-
19:16	<b>ENDPOINT</b> : 中断端点编号	读写	0x0
15:7	保留。	-	-
6:0	<b>ADDRESS</b> : 设备地址	读写	0x00

## USB: MAIN\_CTRL 寄存器

偏移量: 0x040

### 描述

主控制寄存器

表1198。  
MAIN\_CTRL 寄存器

位	描述	类型	复位
31	<b>SIM_TIMING</b> : 模拟时序缩短	读写	0x0
30:3	保留。	-	-
2	<b>PHY_ISO</b> : 控制器上电后隔离 USB 物理层 软件配置控制器后，解除隔离 非隔离 = 0，隔离 = 1	读写	0x1
1	<b>HOST_NDEVICE</b> : 设备模式 = 0，主机模式 = 1	读写	0x0
0	<b>CONTROLLER_EN</b> : 启用控制器	读写	0x0

## USB: SOF\_WR 寄存器

偏移量: 0x044

### 说明

在主机控制器中设置SOF (帧起始) 帧编号。SOF包每1毫秒发送一次，主机会每次将帧编号递增1。

表 1199. SOF\_WR  
寄存器

位	描述	类型	复位
31:11	保留。	-	-
10:0	计数	WF	0x000

## USB: SOF\_RD 寄存器

偏移量: 0x048

### 说明

读取最近一次检测到的SOF (帧开始) 帧编号。设备模式下，接收到的主机最后一个SOF帧编号。主机模式下，主机发送的最后一个SOF帧编号。

表 1200. SOF\_RD  
寄存器

位	描述	类型	复位
31:11	保留。	-	-
10:0	<b>计数</b>	只读	0x000

## USB: SIE\_CTRL 寄存器

偏移量: 0x04c

### 描述

SIE控制寄存器

表 1201. SIE\_CTRL  
寄存器

位	描述	类型	复位
31	<b>EPO_INT_STALL</b> : 设备端: 当EP0发送STALL时, 在EP_STATUS_STALL_NAK中设置位	读写	0x0
30	<b>EPO_DOUBLE_BUF</b> : 设备端: EP0单缓冲 = 0, 双缓冲 = 1	读写	0x0
29	<b>EPO_INT_1BUF</b> : 设备端: 每完成一个EP0缓冲区时, 在BUFF_STATUS中设置位	读写	0x0
28	<b>EPO_INT_2BUF</b> : 设备端: 每完成两个EP0缓冲区时, 在BUFF_STATUS中设置位	读写	0x0
27	<b>EPO_INT_NAK</b> : 设备: 当EP0发送NAK时, 在EP_STATUS_STALL_NAK中置位	读写	0x0
26	<b>DIRECT_EN</b> : 直接总线驱动使能	读写	0x0
25	<b>DIRECT_DP</b> : 直接控制DP	读写	0x0
24	<b>DIRECT_DM</b> : 直接控制DM	读写	0x0
23:20	保留。	-	-
19	<b>EPO_STOP_ON_SHORT_PACKET</b> : 设备: 短包时停止EP0。	读写	0x0
18	<b>TRANSCEIVER_PD</b> : 总线收发器断电	读写	0x0
17	<b>RPU_OPT</b> : 设备: 上拉电阻强度 (0=1K2, 1=2k3)	读写	0x0
16	<b>PULLUP_EN</b> : 设备: 使能上拉电阻	读写	0x0
15	<b>PULLDOWN_EN</b> : 主机: 使能下拉电阻	读写	0x1
14	保留。	-	-
13	<b>RESET_BUS</b> : 主机: 复位总线	SC	0x0
12	<b>RESUME</b> : 设备: 远程唤醒。设备可在挂起后主动发起恢复操作。	SC	0x0
11	<b>VBUS_EN</b> : 主机: 使能VBUS	读写	0x0
10	<b>KEEP_ALIVE_EN</b> : 主机: 使能保活包 (用于低速总线)	读写	0x0
9	<b>SOF_EN</b> : 主机: 使能SOF生成 (用于全速总线)	读写	0x0
8	<b>SOF_SYNC</b> : 主机: 在SOF之后延迟数据包	读写	0x0
7	保留。	-	-
6	<b>PREAMBLE_EN</b> : 主机: 在FS集线器上的LS设备启用前导码	读写	0x0
5	保留。	-	-
4	<b>STOP_TRANS</b> : 主机: 停止事务	SC	0x0

位	描述	类型	复位
3	<b>RECEIVE_DATA</b> : 主机: 接收事务 (IN传输到主机)	读写	0x0
2	<b>SEND_DATA</b> : 主机: 发送事务 (OUT传输来自主机)	读写	0x0
1	<b>SEND_SETUP</b> : 主机: 发送设置数据包	读写	0x0
0	<b>START_TRANS</b> : 主机: 启动事务	SC	0x0

## USB: SIE\_STATUS寄存器

偏移量: 0x050

### 描述

SIE状态寄存器

表1202。  
SIE\_STATUS寄存器

位	描述	类型	复位
31	<b>DATA_SEQ_ERROR</b> : 数据序列错误。  设备在以下情况下可能触发序列错误：  * 收到SETUP包后紧跟DATA1包 (数据阶段应始终为DATA0) * 主机收到的OUT包数据PID与从DPSRAM读取的缓冲区控制寄存器中的数据PID不匹配  主机在以下情况下可能触发数据序列错误：  * 设备发送的IN包数据PID错误	WC	0x0
30	<b>ACK_REC</b> : 已接收ACK。由主机和设备共同引发。	WC	0x0
29	<b>STALL_REC</b> : 主机: 接收到STALL信号	WC	0x0
28	<b>NAK_REC</b> : 主机: 接收到NAK信号	WC	0x0
27	<b>RX_TIMEOUT</b> : 若在USB规范规定的最大时间内未收到ACK，则主机和设备均会引发接收超时 (RX timeout)。	WC	0x0
26	<b>RX_OVERFLOW</b> : 若串行接收引擎接收数据过快，则会引发接收溢出。	WC	0x0
25	<b>BIT_STUFF_ERROR</b> : 位填充错误。由串行接收引擎检测并引发。	WC	0x0
24	<b>CRC_ERROR</b> : CRC错误。由串行接收引擎检测并引发。	WC	0x0
23	<b>ENDPOINT_ERROR</b> : 端点发生错误。请读取ep_rx_error和ep_tx_error寄存器以确定具体的错误端点。	WC	0x0
22:20	保留。	-	-
19	<b>BUS_RESET</b> : 设备: 接收到总线复位信号	WC	0x0

位	描述	类型	复位
18	<b>TRANS_COMPLETE</b> : 事务完成。  由设备引发，条件如下：  * 当IN或OUT数据包在缓冲控制寄存器中设置了LAST_BUFF位时发送  由主机引发，条件如下：  * 发送setup数据包但后续无数据输入或输出事务* 接收到IN数据包且缓冲控制寄存器中设置了LAST_BUFF位* 接收到长度为零的IN数据包* 发送OUT数据包且缓冲控制寄存器中设置了LAST_BUFF位	WC	0x0
17	<b>SETUP_REC</b> : 设备：接收到设置包	WC	0x0
16	<b>CONNECTED</b> : 设备：已连接	只读	0x0
15:13	保留。	-	-
12	<b>RX_SHORT_PACKET</b> : 设备或主机已接收短包。当接收的数据少于缓冲区控制寄存器中配置的值时，会发生此情况。设备： 如果设备使用双缓冲模式，写回状态到缓冲区控制寄存器后，缓冲区选择不会切换。此措施用于防止用户重置缓冲区控制寄存器前，端点上进行任何后续事务。主机：当前传输将被提前终止。	WC	0x0
11	<b>RESUME</b> : 主机：设备已发起远程恢复。设备：主机已发起恢复操作。	WC	0x0
10	<b>VBUS_OVER_CURR</b> : 检测到VBUS过流	只读	0x0
9:8	<b>SPEED</b> : 主机：设备速度。断开连接 = 00，低速 = 01，全速 = 10	只读	0x0
7:5	保留。	-	-
4	<b>SUSPENDED</b> : 总线处于挂起状态。适用于设备和主机。如果未启用Keep Alive或SOF帧，主机和设备将进入挂起状态。	只读	0x0
3:2	<b>LINE_STATE</b> : USB总线线路状态	只读	0x0
1	保留。	-	-
0	<b>VBUS_DETECTED</b> : 设备：检测到VBUS	只读	0x0

## USB: INT\_EP\_CTRL寄存器

偏移: 0x054

### 描述

中断端点控制寄存器

表1203。  
INT\_EP\_CTRL寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:1	<b>INT_EP_ACTIVE</b> : 主机：启用中断端点1 → 15	读写	0x0000
0	保留。	-	-

## USB: BUFF\_STATUS寄存器

偏移: 0x058

**描述**

缓冲区状态寄存器。此位被设置表示该端点的缓冲区已完成（前提是已启用缓冲区中断）。可能会同时完成两个缓冲区，因此清除缓冲区状态位后，该位可能在下一个时钟周期立即重新被置位。

表1204。  
BUFF\_STATUS  
寄存器

位	描述	类型	复位
31	<b>EP15_OUT</b>	WC	0x0
30	<b>EP15_IN</b>	WC	0x0
29	<b>EP14_OUT</b>	WC	0x0
28	<b>EP14_IN</b>	WC	0x0
27	<b>EP13_OUT</b>	WC	0x0
26	<b>EP13_IN</b>	WC	0x0
25	<b>EP12_OUT</b>	WC	0x0
24	<b>EP12_IN</b>	WC	0x0
23	<b>EP11_OUT</b>	WC	0x0
22	<b>EP11_IN</b>	WC	0x0
21	<b>EP10_OUT</b>	WC	0x0
20	<b>EP10_IN</b>	WC	0x0
19	<b>EP9_OUT</b>	WC	0x0
18	<b>EP9_IN</b>	WC	0x0
17	<b>EP8_OUT</b>	WC	0x0
16	<b>EP8_IN</b>	WC	0x0
15	<b>EP7_OUT</b>	WC	0x0
14	<b>EP7_IN</b>	WC	0x0
13	<b>EP6_OUT</b>	WC	0x0
12	<b>EP6_IN</b>	WC	0x0
11	<b>EP5_OUT</b>	WC	0x0
10	<b>EP5_IN</b>	WC	0x0
9	<b>EP4_OUT</b>	WC	0x0
8	<b>EP4_IN</b>	WC	0x0
7	<b>EP3_OUT</b>	WC	0x0
6	<b>EP3_IN</b>	WC	0x0
5	<b>EP2_OUT</b>	WC	0x0
4	<b>EP2_IN</b>	WC	0x0
3	<b>EP1_OUT</b>	WC	0x0
2	<b>EP1_IN</b>	WC	0x0
1	<b>EP0_OUT</b>	WC	0x0
0	<b>EP0_IN</b>	WC	0x0

## USB: BUFF\_CPU\_SHOULD\_HANDLE寄存器

偏移: 0x05c

### 描述

应处理哪一个双缓冲区。仅在每个缓冲区产生中断时有效（即非每两个缓冲区产生一次中断）。对主机中断端点轮询无效，因为它们仅为单缓冲。

表1205。  
BUFF\_CPU\_SHOULD\_HANDLE寄存器

位	描述	类型	复位
31	<b>EP15_OUT</b>	只读	0x0
30	<b>EP15_IN</b>	只读	0x0
29	<b>EP14_OUT</b>	只读	0x0
28	<b>EP14_IN</b>	只读	0x0
27	<b>EP13_OUT</b>	只读	0x0
26	<b>EP13_IN</b>	只读	0x0
25	<b>EP12_OUT</b>	只读	0x0
24	<b>EP12_IN</b>	只读	0x0
23	<b>EP11_OUT</b>	只读	0x0
22	<b>EP11_IN</b>	只读	0x0
21	<b>EP10_OUT</b>	只读	0x0
20	<b>EP10_IN</b>	只读	0x0
19	<b>EP9_OUT</b>	只读	0x0
18	<b>EP9_IN</b>	只读	0x0
17	<b>EP8_OUT</b>	只读	0x0
16	<b>EP8_IN</b>	只读	0x0
15	<b>EP7_OUT</b>	只读	0x0
14	<b>EP7_IN</b>	只读	0x0
13	<b>EP6_OUT</b>	只读	0x0
12	<b>EP6_IN</b>	只读	0x0
11	<b>EP5_OUT</b>	只读	0x0
10	<b>EP5_IN</b>	只读	0x0
9	<b>EP4_OUT</b>	只读	0x0
8	<b>EP4_IN</b>	只读	0x0
7	<b>EP3_OUT</b>	只读	0x0
6	<b>EP3_IN</b>	只读	0x0
5	<b>EP2_OUT</b>	只读	0x0
4	<b>EP2_IN</b>	只读	0x0
3	<b>EP1_OUT</b>	只读	0x0
2	<b>EP1_IN</b>	只读	0x0
1	<b>EP0_OUT</b>	只读	0x0

位	描述	类型	复位
0	EP0_IN	只读	0x0

## USB：EP\_ABORT寄存器

偏移：0x060

### 描述

仅设备端：可设置为忽略该端点的缓冲区控制寄存器，以便撤销缓冲区。在此位被清除之前，每次访问该端点均会发送NAK。当EP\_ABORT\_DONE中相应位被置位时，表示可以安全修改缓冲区控制寄存器。

表1206。  
EP\_ABORT 寄存器

位	描述	类型	复位
31	EP15_OUT	读写	0x0
30	EP15_IN	读写	0x0
29	EP14_OUT	读写	0x0
28	EP14_IN	读写	0x0
27	EP13_OUT	读写	0x0
26	EP13_IN	读写	0x0
25	EP12_OUT	读写	0x0
24	EP12_IN	读写	0x0
23	EP11_OUT	读写	0x0
22	EP11_IN	读写	0x0
21	EP10_OUT	读写	0x0
20	EP10_IN	读写	0x0
19	EP9_OUT	读写	0x0
18	EP9_IN	读写	0x0
17	EP8_OUT	读写	0x0
16	EP8_IN	读写	0x0
15	EP7_OUT	读写	0x0
14	EP7_IN	读写	0x0
13	EP6_OUT	读写	0x0
12	EP6_IN	读写	0x0
11	EP5_OUT	读写	0x0
10	EP5_IN	读写	0x0
9	EP4_OUT	读写	0x0
8	EP4_IN	读写	0x0
7	EP3_OUT	读写	0x0
6	EP3_IN	读写	0x0
5	EP2_OUT	读写	0x0
4	EP2_IN	读写	0x0

位	描述	类型	复位
3	<b>EP1_OUT</b>	读写	0x0
2	<b>EP1_IN</b>	读写	0x0
1	<b>EP0_OUT</b>	读写	0x0
0	<b>EP0_IN</b>	读写	0x0

## USB：EP\_ABORT\_DONE 寄存器

偏移: 0x064

### 描述

仅设备端：与 EP\_ABORT 配合使用。端点空闲时设置，以告知程序员可安全修改缓冲区控制寄存器。

表1207。  
EP\_ABORT\_DONE  
寄存器

位	描述	类型	复位
31	<b>EP15_OUT</b>	WC	0x0
30	<b>EP15_IN</b>	WC	0x0
29	<b>EP14_OUT</b>	WC	0x0
28	<b>EP14_IN</b>	WC	0x0
27	<b>EP13_OUT</b>	WC	0x0
26	<b>EP13_IN</b>	WC	0x0
25	<b>EP12_OUT</b>	WC	0x0
24	<b>EP12_IN</b>	WC	0x0
23	<b>EP11_OUT</b>	WC	0x0
22	<b>EP11_IN</b>	WC	0x0
21	<b>EP10_OUT</b>	WC	0x0
20	<b>EP10_IN</b>	WC	0x0
19	<b>EP9_OUT</b>	WC	0x0
18	<b>EP9_IN</b>	WC	0x0
17	<b>EP8_OUT</b>	WC	0x0
16	<b>EP8_IN</b>	WC	0x0
15	<b>EP7_OUT</b>	WC	0x0
14	<b>EP7_IN</b>	WC	0x0
13	<b>EP6_OUT</b>	WC	0x0
12	<b>EP6_IN</b>	WC	0x0
11	<b>EP5_OUT</b>	WC	0x0
10	<b>EP5_IN</b>	WC	0x0
9	<b>EP4_OUT</b>	WC	0x0
8	<b>EP4_IN</b>	WC	0x0
7	<b>EP3_OUT</b>	WC	0x0
6	<b>EP3_IN</b>	WC	0x0

位	描述	类型	复位
5	<b>EP2_OUT</b>	WC	0x0
4	<b>EP2_IN</b>	WC	0x0
3	<b>EP1_OUT</b>	WC	0x0
2	<b>EP1_IN</b>	WC	0x0
1	<b>EP0_OUT</b>	WC	0x0
0	<b>EP0_IN</b>	WC	0x0

## USB: EP\_STALL\_ARM 寄存器

偏移: 0x068

### 描述

设备：此位必须与缓冲区控制寄存器中的 **STALL** 位同时设置，方可向 EP0 上发送 STALL。

设备控制器在收到 SETUP 包时会清除这些位，因 USB 规范要求收到 SETUP 包时必须清除 STALL 状态。

表1208。  
EP\_STALL\_ARM 寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>EP0_OUT</b>	读写	0x0
0	<b>EP0_IN</b>	读写	0x0

## USB: NAK\_POLL 寄存器

偏移: 0x06C

### 描述

由主机控制器使用。设置设备回复 NAK 时重试前的等待时间，单位为微秒。

表1209。  
NAK\_POLL 寄存器

位	描述	类型	复位
31:28	<b>RETRY_COUNT_HI</b> : nak_retry 计数的第9至第6位	只读	0x0
27	<b>EPX_STOPPED_ON_NAK</b> : EPX 因接收到 nak 而停止轮询	WC	0x0
26	<b>STOP_EPX_ON_NAK</b> : 收到 NAK 时停止轮询 epx	读写	0x0
25:16	<b>DELAY_FS</b> : 全速设备的 NAK 轮询间隔	读写	0x010
15:10	<b>RETRY_COUNT_LO</b> : nak_retry_count 的第5至0位	只读	0x00
9:0	<b>DELAY_LS</b> : 低速设备的 NAK 轮询间隔	读写	0x010

## USB: EP\_STATUS\_STALL\_NAK 寄存器

偏移: 0x070

### 描述

设备：当 **IRQ\_ON\_NAK** 或 **IRQ\_ON\_STALL** 位被设置时，该位也被置位。对于 EP0，该信号源自 **SIE\_CTRL**。对于所有其他端点，该信号来源于端点控制寄存器。

表1210。  
EP\_STATUS\_STALL\_NAK 寄存器

位	描述	类型	复位
31	<b>EP15_OUT</b>	WC	0x0
30	<b>EP15_IN</b>	WC	0x0

位	描述	类型	复位
29	<b>EP14_OUT</b>	WC	0x0
28	<b>EP14_IN</b>	WC	0x0
27	<b>EP13_OUT</b>	WC	0x0
26	<b>EP13_IN</b>	WC	0x0
25	<b>EP12_OUT</b>	WC	0x0
24	<b>EP12_IN</b>	WC	0x0
23	<b>EP11_OUT</b>	WC	0x0
22	<b>EP11_IN</b>	WC	0x0
21	<b>EP10_OUT</b>	WC	0x0
20	<b>EP10_IN</b>	WC	0x0
19	<b>EP9_OUT</b>	WC	0x0
18	<b>EP9_IN</b>	WC	0x0
17	<b>EP8_OUT</b>	WC	0x0
16	<b>EP8_IN</b>	WC	0x0
15	<b>EP7_OUT</b>	WC	0x0
14	<b>EP7_IN</b>	WC	0x0
13	<b>EP6_OUT</b>	WC	0x0
12	<b>EP6_IN</b>	WC	0x0
11	<b>EP5_OUT</b>	WC	0x0
10	<b>EP5_IN</b>	WC	0x0
9	<b>EP4_OUT</b>	WC	0x0
8	<b>EP4_IN</b>	WC	0x0
7	<b>EP3_OUT</b>	WC	0x0
6	<b>EP3_IN</b>	WC	0x0
5	<b>EP2_OUT</b>	WC	0x0
4	<b>EP2_IN</b>	WC	0x0
3	<b>EP1_OUT</b>	WC	0x0
2	<b>EP1_IN</b>	WC	0x0
1	<b>EP0_OUT</b>	WC	0x0
0	<b>EP0_IN</b>	WC	0x0

## USB: USB\_MUXING寄存器

偏移: 0x074

### 描述

USB控制器的连接位置。默认应设置为to\_phy。

表1211。  
USB\_MUXING寄存器

位	描述	类型	复位
31	<b>SWAP_DPDM</b> : 交换USB PHY的DP和DM引脚及所有相关控制，并翻转接收的差分信号。可用于切换PCB上的USB DP/DM。 此操作在底层完成，覆盖所有其他控制。	读写	0x0
30:5	保留。	-	-
4	<b>USBPHY_AS_GPIO</b> : 将USB DP和DM引脚用作GPIO，而非连接到USB控制器。	读写	0x0
3	<b>SOFTCON</b>	读写	0x0
2	<b>TO_DIGITAL_PAD</b>	读写	0x0
1	<b>TO_EXTPHY</b>	读写	0x0
0	<b>TO_PHY</b>	读写	0x1

## USB: USB\_PWR 寄存器

偏移量: 0x078

### 说明

当VBUS信号未接入GPIO时，对电源信号进行覆盖。先设置覆盖值，再启用覆盖，切换至覆盖值。

表1212. USB\_PWR  
寄存器

位	描述	类型	复位
31:6	保留。	-	-
5	<b>OVERRCURR_DETECT_EN</b>	读写	0x0
4	<b>OVERRCURR_DETECT</b>	读写	0x0
3	<b>VBUS_DETECT_OVERRIDE_EN</b>	读写	0x0
2	<b>VBUS_DETECT</b>	读写	0x0
1	<b>VBUS_EN_OVERRIDE_EN</b>	读写	0x0
0	<b>VBUS_EN</b>	读写	0x0

## USB: USBPHY\_DIRECT 寄存器

偏移量: 0x07C

### 说明

此寄存器允许对USB物理层直接控制。需与usbphy\_direct\_override寄存器配合使用以启用相应覆盖位。

表1213.  
USBPHY\_DIRECT  
寄存器

位	描述	类型	复位
31:26	保留。	-	-
25	<b>RX_DM_OVERRIDE</b> : 覆盖 rx_dm 值至控制器	读写	0x0
24	<b>RX_DP_OVERRIDE</b> : 覆盖 rx_dp 值至控制器	读写	0x0
23	<b>RX_DD_OVERRIDE</b> : 覆盖 rx_dd 值至控制器	读写	0x0
22	<b>DM_OVV</b> : DM 过压	只读	0x0
21	<b>DP_OVV</b> : DP 过压	只读	0x0
20	<b>DM_OVCN</b> : DM 过流	只读	0x0

位	描述	类型	复位
19	<b>DP_OVCN</b> : DP 过流	只读	0x0
18	<b>RX_DM</b> : DPM 引脚状态	只读	0x0
17	<b>RX_DP</b> : DPP 引脚状态	只读	0x0
16	<b>RX_DD</b> : 差分 RX	只读	0x0
15	<b>TX_DIFFMODE</b> : TX_DIFFMODE=0: 单端模式 TX_DIFFMODE=1: 差分驱动模式 (TX_DM, TX_DM_OE 被忽略)	读写	0x0
14	<b>TX_FSSLEW</b> : TX_FSSLEW=0: 低速转换率 TX_FSSLEW=1: 全速转换率	读写	0x0
13	<b>TX_PD</b> : TX 电源关闭覆盖 (若启用覆盖)。1=断电状态。	读写	0x0
12	<b>RX_PD</b> : RX 电源关闭覆盖 (若启用覆盖)。1=断电状态。	读写	0x0
11	<b>TX_DM</b> : 输出数据。TX_DIFFMODE=1 时忽略。 TX_DIFFMODE=0 时, 仅驱动 DPM。TX_DM_OE=1 时启用驱动。 DPM=TX_DM	读写	0x0
10	<b>TX_DP</b> : 输出数据。若 TX_DIFFMODE=1, 则驱动 DPP/DPM 差分对。 TX_DP_OE=1 时启用驱动。DPP=TX_DP, DPM=~TX_DP。 若 TX_DIFFMODE=0, 则仅驱动 DPP。TX_DP_OE=1: 启用驱动。 DPP=TX_DP	读写	0x0
9	<b>TX_DM_OE</b> : 输出使能。当 TX_DIFFMODE=1 时, 忽略该项。 当 TX_DIFFMODE=0 时, 仅对 DPM 使能 OE。0—DPM 处于高阻状态; 1—DPM 驱动。	读写	0x0
8	<b>TX_DP_OE</b> : 输出使能。当 TX_DIFFMODE=1 时, DPP/DPM 差分对使能 OE。 0—DPP/DPM 处于高阻状态; 1—DPP/DPM 驱动。 当 TX_DIFFMODE=0 时, 仅对 DPP 使能 OE。0—DPP 处于高阻状态; 1—DPP 驱动。	读写	0x0
7	保留。	-	-
6	<b>DM_PULLDN_EN</b> : DM 下拉使能。	读写	0x0
5	<b>DM_PULLUP_EN</b> : DM 上拉使能。	读写	0x0
4	<b>DM_PULLUP_HISEL</b> : 启用第二个 DM 上拉电阻。0—上拉电阻为 Rpu2; 1—上拉电阻为 Rpu1 + Rpu2。	读写	0x0
3	保留。	-	-
2	<b>DP_PULLDN_EN</b> : DP 下拉使能	读写	0x0
1	<b>DP_PULLUP_EN</b> : DP 上拉使能	读写	0x0
0	<b>DP_PULLUP_HISEL</b> : 启用第二个 DP 上拉电阻。0 - 下拉 = Rpu2; 1 - 下拉 = Rpu1 + Rpu2	读写	0x0

## USB: USBPHY\_DIRECT\_OVERRIDE 寄存器

偏移: 0x080

### 描述

usbphy\_direct 中各控制项的覆盖使能

表 1214。  
USBPHY\_DIRECT\_OVERRIDE 寄存器

位	描述	类型	复位
31:19	保留。	-	-
18	<b>RX_DM_OVERRIDE_EN</b>	读写	0x0

位	描述	类型	复位
17	<b>RX_DP_OVERRIDE_EN</b>	读写	0x0
16	<b>RX_DD_OVERRIDE_EN</b>	读写	0x0
15	<b>TX_DIFFMODE_OVERRIDE_EN</b>	读写	0x0
14:13	保留。	-	-
12	<b>DM_PULLUP_OVERRIDE_EN</b>	读写	0x0
11	<b>TX_FSSLEW_OVERRIDE_EN</b>	读写	0x0
10	<b>TX_PD_OVERRIDE_EN</b>	读写	0x0
9	<b>RX_PD_OVERRIDE_EN</b>	读写	0x0
8	<b>TX_DM_OVERRIDE_EN</b>	读写	0x0
7	<b>TX_DP_OVERRIDE_EN</b>	读写	0x0
6	<b>TX_DM_OE_OVERRIDE_EN</b>	读写	0x0
5	<b>TX_DP_OE_OVERRIDE_EN</b>	读写	0x0
4	<b>DM_PULLDN_EN_OVERRIDE_EN</b>	读写	0x0
3	<b>DP_PULLDN_EN_OVERRIDE_EN</b>	读写	0x0
2	<b>DP_PULLUP_EN_OVERRIDE_EN</b>	读写	0x0
1	<b>DM_PULLUP_HISEL_OVERRIDE_EN</b>	读写	0x0
0	<b>DP_PULLUP_HISEL_OVERRIDE_EN</b>	读写	0x0

## USB: USBPHY\_TRIM 寄存器

偏移: 0x084

### 描述

用于调整 USB 物理层下拉电阻的微调值。

表 1215。  
USBPHY\_TRIM  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12:8	<b>DM_PULLDN_TRIM:</b> 驱动至 USB PHY 的值 DM 下拉电阻微调控制 实验数据表明复位值有效，但该寄存器允许根据需要进行调整	读写	0x1f
7:5	保留。	-	-
4:0	<b>DP_PULLDN_TRIM:</b> 驱动至 USB PHY 的值 DP 下拉电阻微调控制 实验数据表明复位值有效，但该寄存器允许根据需要进行调整	读写	0x1f

## USB: LINESTATE\_TUNING 寄存器

偏移: 0x088

### 描述

仅供调试使用。

表 1216。  
LINESTATE\_TUNING  
寄存器

位	描述	类型	复位
31:12	保留。	-	-
11:8	<b>SPARE_FIX</b>	读写	0x0
7	<b>DEV_LS_WAKE_FIX</b> : 设备 - 在任何非空闲信号时退出挂起状态，且不受1毫秒定时器限制	读写	0x1
6	<b>DEV_RX_ERR QUIESCE</b> : 设备 - 抑制重复错误，直到设备状态机下一次进入解码入站数据包的阶段。	读写	0x1
5	<b>SIE_RX_CHATTER_SEO_FIX</b> : 接收 - 从线路抖动或比特填充错误恢复时，将SEO视为抖动结束标志，同时视为8个连续空闲位	读写	0x1
4	<b>SIE_RX_BITSTUFF_FIX</b> : 接收 - 当rx_dasm报告比特填充错误时，无条件终止接收解码，以避免在特定数据包阶段出现挂起。	读写	0x1
3	<b>DEV_BUFF_CONTROL_DOUBLE_READ_FIX</b> : 设备 - 控制器状态机对缓冲区状态存储地址执行两次读取，以避免采样亚稳态数据。仅当两次读取结果一致时，启用的缓冲区方可使用。	读写	0x1
2	<b>MULTI_HUB_FIX</b> : 主机 - 增加数据包间隔和切换超时时间，以适应最坏情况下集线器的延迟。	读写	0x0
1	<b>LINESTATE_DELAY</b> : 设备/主机 - 对线路状态采样增加额外的1位硬件消抖。	读写	0x0
0	<b>RCV_DELAY</b> : 设备 - 注册接收数据以补偿集线器结束标志(EOP)前的位漂移。仅影响特定集线器。	读写	0x0

## USB：INTR寄存器

偏移：0x08c

### 描述

原始中断

表 1217. INTR  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23	<b>EPX_STOPPED_ON_NAK</b> : 来源：NAK_POLL.EPX_STOPPED_ON_NAK	只读	0x0
22	<b>DEV_SM_WATCHDOG_FIRED</b> : 来源：DEV_SM_WATCHDOG.FIRED	只读	0x0
21	<b>ENDPOINT_ERROR</b> : 来源：SIE_STATUS.ENDPOINT_ERROR	只读	0x0
20	<b>RX_SHORT_PACKET</b> : 来源：SIE_STATUS.RX_SHORT_PACKET	只读	0x0
19	<b>EP_STALL_NAK</b> : 当EP_STATUS_STALL_NAK中任一位被置位时触发。通过清除EP_STATUS_STALL_NAK的所有位来复位。	只读	0x0
18	<b>ABORT_DONE</b> : 当ABORT_DONE中任一位被置位时触发。通过清除ABORT_DONE的所有位来复位。	只读	0x0
17	<b>DEV_SOF</b> : 每当设备接收 SOF (帧起始) 数据包时置位。通过读取 SOF_RD 清除	只读	0x0
16	<b>SETUP_REQ</b> : 设备。来源：SIE_STATUS.SETUP_REC	只读	0x0
15	<b>DEV_RESUME_FROM_HOST</b> : 设备接收到来自主机的恢复信号时置位。通过写入 SIE_STATUS.RESUME 清除	只读	0x0

位	描述	类型	复位
14	<b>DEV_SUSPEND</b> : 设备挂起状态发生变化时置位。通过写入 SIE_STATUS.SUSPENDED 清除	只读	0x0
13	<b>DEV_CONN_DIS</b> : 设备连接状态发生变化时置位。通过写入 SIE_STATUS.CONNECTED 清除	只读	0x0
12	<b>BUS_RESET</b> : 来源: SIE_STATUS.BUS_RESET	只读	0x0
11	<b>VBUS_DETECT</b> : 来源: SIE_STATUS.VBUS_DETECTED	只读	0x0
10	<b>STALL</b> : 来源: SIE_STATUSSTALL_REC	只读	0x0
9	<b>ERROR_CRC</b> : 来源: SIE_STATUS.CRC_ERROR	只读	0x0
8	<b>ERROR_BIT_STUFF</b> : 来源: SIE_STATUS.BIT_STUFF_ERROR	只读	0x0
7	<b>ERROR_RX_OVERFLOW</b> : 来源: SIE_STATUS.RX_OVERFLOW	只读	0x0
6	<b>ERROR_RX_TIMEOUT</b> : 来源: SIE_STATUS.RX_TIMEOUT	只读	0x0
5	<b>ERROR_DATA_SEQ</b> : 来源: SIE_STATUS.DATA_SEQ_ERROR	只读	0x0
4	<b>BUFF_STATUS</b> : 当BUFF_STATUS中任意位被置位时触发。通过清除BUFF_STATUS的所有位来复位。	只读	0x0
3	<b>TRANS_COMPLETE</b> : 每当SIE_STATUS.TRANS_COMPLETE被置位时触发。通过写入该位来复位。	只读	0x0
2	<b>HOST_SOF</b> : 主机——每当主机发送SOF (帧起始标志) 时触发。通过读取 SOF_RD 清除	只读	0x0
1	<b>HOST_RESUME</b> : 主机——当设备唤醒主机时触发。通过写入 SIE_STATUS.RESUME 清除	只读	0x0
0	<b>HOST_CONN_DIS</b> : 主机——当设备连接或断开 (即SIE_STATUS.SPEED发生变化) 时触发。通过写入SIE_STATUS.SPEED来复位	只读	0x0

## USB: INTE 寄存器

偏移: 0x090

### 描述

中断使能

表1218. INTE 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23	<b>EPX_STOPPED_ON_NAK</b> : 来源: NAK_POLL.EPX_STOPPED_ON_NAK	读写	0x0
22	<b>DEV_SM_WATCHDOG_FIRED</b> : 来源: DEV_SM_WATCHDOG.FIRED	读写	0x0
21	<b>ENDPOINT_ERROR</b> : 来源: SIE_STATUS.ENDPOINT_ERROR	读写	0x0
20	<b>RX_SHORT_PACKET</b> : 来源: SIE_STATUS.RX_SHORT_PACKET	读写	0x0
19	<b>EP_STALL_NAK</b> : 当EP_STATUS_STALL_NAK中任一位被置位时触发。通过清除EP_STATUS_STALL_NAK的所有位来复位。	读写	0x0
18	<b>ABORT_DONE</b> : 当ABORT_DONE中任一位被置位时触发。通过清除ABORT_DONE的所有位来复位。	读写	0x0
17	<b>DEV_SOF</b> : 每当设备接收 SOF (帧起始) 数据包时置位。通过读取 SOF_RD 清除	读写	0x0

位	描述	类型	复位
16	<b>SETUP_REQ</b> : 设备。来源: SIE_STATUS.SETUP_REC	读写	0x0
15	<b>DEV_RESUME_FROM_HOST</b> : 设备接收到来自主机的恢复信号时置位。通过写入 SIE_STATUS.RESUME 清除	读写	0x0
14	<b>DEV_SUSPEND</b> : 设备挂起状态发生变化时置位。通过写入 SIE_STATUS.SUSPENDED 清除	读写	0x0
13	<b>DEV_CONN_DIS</b> : 设备连接状态发生变化时置位。通过写入 SIE_STATUS.CONNNECTED 清除	读写	0x0
12	<b>BUS_RESET</b> : 来源: SIE_STATUS.BUS_RESET	读写	0x0
11	<b>VBUS_DETECT</b> : 来源: SIE_STATUS.VBUS_DETECTED	读写	0x0
10	<b>STALL</b> : 来源: SIE_STATUSSTALL_REC	读写	0x0
9	<b>ERROR_CRC</b> : 来源: SIE_STATUS.CRC_ERROR	读写	0x0
8	<b>ERROR_BIT_STUFF</b> : 来源: SIE_STATUS.BIT_STUFF_ERROR	读写	0x0
7	<b>ERROR_RX_OVERFLOW</b> : 来源: SIE_STATUS.RX_OVERFLOW	读写	0x0
6	<b>ERROR_RX_TIMEOUT</b> : 来源: SIE_STATUS.RX_TIMEOUT	读写	0x0
5	<b>ERROR_DATA_SEQ</b> : 来源: SIE_STATUS.DATA_SEQ_ERROR	读写	0x0
4	<b>BUFF_STATUS</b> : 当BUFF_STATUS中任意位被置位时触发。通过清除BUFF_STATUS的所有位来复位。	读写	0x0
3	<b>TRANS_COMPLETE</b> : 每当SIE_STATUS.TRANS_COMPLETE被置位时触发。通过写入该位来复位。	读写	0x0
2	<b>HOST_SOF</b> : 主机——每当主机发送SOF (帧起始标志) 时触发。通过读取 SOF_RD 清除	读写	0x0
1	<b>HOST_RESUME</b> : 主机——当设备唤醒主机时触发。通过写入 SIE_STATUS.RESUME 清除	读写	0x0
0	<b>HOST_CONN_DIS</b> : 主机——当设备连接或断开 (即SIE_STATUS.SPEED发生变化) 时触发。通过写入SIE_STATUS.SPEED来复位	读写	0x0

## USB: INTF 寄存器

偏移: 0x094

### 描述

中断强制

表1219. INTF 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23	<b>EPX_STOPPED_ON_NAK</b> : 来源: NAK_POLL.EPX_STOPPED_ON_NAK	读写	0x0
22	<b>DEV_SM_WATCHDOG_FIRED</b> : 来源: DEV_SM_WATCHDOG.FIRED	读写	0x0
21	<b>ENDPOINT_ERROR</b> : 来源: SIE_STATUS.ENDPOINT_ERROR	读写	0x0
20	<b>RX_SHORT_PACKET</b> : 来源: SIE_STATUS.RX_SHORT_PACKET	读写	0x0
19	<b>EP_STALL_NAK</b> : 当EP_STATUS_STALL_NAK中任一位被置位时触发。通过清除EP_STATUS_STALL_NAK的所有位来复位。	读写	0x0

位	描述	类型	复位
18	<b>ABORT_DONE</b> : 当ABORT_DONE中任一位被置位时触发。通过清除ABORT_DONE的所有位来复位。	读写	0x0
17	<b>DEV_SOF</b> : 每当设备接收 SOF (帧起始) 数据包时置位。 通过读取 SOF_RD 清除	读写	0x0
16	<b>SETUP_REQ</b> : 设备。来源: SIE_STATUS.SETUP_REC	读写	0x0
15	<b>DEV_RESUME_FROM_HOST</b> : 设备接收到来自主机的恢复信号时置位。通过写入 SIE_STATUS.RESUME 清除	读写	0x0
14	<b>DEV_SUSPEND</b> : 设备挂起状态发生变化时置位。通过写入 SIE_STATUS.SUSPENDED 清除	读写	0x0
13	<b>DEV_CONN_DIS</b> : 设备连接状态发生变化时置位。通过写入 SIE_STATUS.CNNECTED 清除	读写	0x0
12	<b>BUS_RESET</b> : 来源: SIE_STATUS.BUS_RESET	读写	0x0
11	<b>VBUS_DETECT</b> : 来源: SIE_STATUS.VBUS_DETECTED	读写	0x0
10	<b>STALL</b> : 来源: SIE_STATUSSTALL_REC	读写	0x0
9	<b>ERROR_CRC</b> : 来源: SIE_STATUS.CRC_ERROR	读写	0x0
8	<b>ERROR_BIT_STUFF</b> : 来源: SIE_STATUS.BIT_STUFF_ERROR	读写	0x0
7	<b>ERROR_RX_OVERFLOW</b> : 来源: SIE_STATUS.RX_OVERFLOW	读写	0x0
6	<b>ERROR_RX_TIMEOUT</b> : 来源: SIE_STATUS.RX_TIMEOUT	读写	0x0
5	<b>ERROR_DATA_SEQ</b> : 来源: SIE_STATUS.DATA_SEQ_ERROR	读写	0x0
4	<b>BUFF_STATUS</b> : 当BUFF_STATUS中任意位被置位时触发。通过清除BUFF_STATUS的所有位来复位。	读写	0x0
3	<b>TRANS_COMPLETE</b> : 每当SIE_STATUS.TRANS_COMPLETE被置位时触发。 通过写入该位来复位。	读写	0x0
2	<b>HOST_SOF</b> : 主机——每当主机发送SOF (帧起始标志) 时触发。 通过读取 SOF_RD 清除	读写	0x0
1	<b>HOST_RESUME</b> : 主机——当设备唤醒主机时触发。通过写入 SIE_STATUS.RESUME 清除	读写	0x0
0	<b>HOST_CONN_DIS</b> : 主机——当设备连接或断开 (即SIE_STATUS.SPEED发生变化) 时触发。通过写入SIE_STATUS.SPEED来复位	读写	0x0

## USB: INTS寄存器

偏移: 0x098

### 说明

掩码与强制后的中断状态

表1220. INTS寄存器

位	描述	类型	复位
31:24	保留。	-	-
23	<b>EPX_STOPPED_ON_NAK</b> : 来源: NAK_POLL.EPX_STOPPED_ON_NAK	只读	0x0
22	<b>DEV_SM_WATCHDOG_FIRED</b> : 来源: DEV_SM_WATCHDOG.FIRED	只读	0x0
21	<b>ENDPOINT_ERROR</b> : 来源: SIE_STATUS.ENDPOINT_ERROR	只读	0x0

位	描述	类型	复位
20	<b>RX_SHORT_PACKET</b> : 来源: SIE_STATUS.RX_SHORT_PACKET	只读	0x0
19	<b>EP_STALL_NAK</b> : 当EP_STATUS_STALL_NAK中任一位被置位时触发。通过清除EP_STATUS_STALL_NAK的所有位来复位。	只读	0x0
18	<b>ABORT_DONE</b> : 当ABORT_DONE中任一位被置位时触发。通过清除ABORT_DONE的所有位来复位。	只读	0x0
17	<b>DEV_SOF</b> : 每当设备接收 SOF (帧起始) 数据包时置位。 通过读取 SOF_RD 清除	只读	0x0
16	<b>SETUP_REQ</b> : 设备。来源: SIE_STATUS.SETUP_REC	只读	0x0
15	<b>DEV_RESUME_FROM_HOST</b> : 设备接收到来自主机的恢复信号时置位。通过写入 SIE_STATUS.RESUME 清除	只读	0x0
14	<b>DEV_SUSPEND</b> : 设备挂起状态发生变化时置位。通过写入 SIE_STATUS.SUSPENDED 清除	只读	0x0
13	<b>DEV_CONN_DIS</b> : 设备连接状态发生变化时置位。通过写入 SIE_STATUS.CONNECTED 清除	只读	0x0
12	<b>BUS_RESET</b> : 来源: SIE_STATUS.BUS_RESET	只读	0x0
11	<b>VBUS_DETECT</b> : 来源: SIE_STATUS.VBUS_DETECTED	只读	0x0
10	<b>STALL</b> : 来源: SIE_STATUSSTALL_REC	只读	0x0
9	<b>ERROR_CRC</b> : 来源: SIE_STATUS.CRC_ERROR	只读	0x0
8	<b>ERROR_BIT_STUFF</b> : 来源: SIE_STATUS.BIT_STUFF_ERROR	只读	0x0
7	<b>ERROR_RX_OVERFLOW</b> : 来源: SIE_STATUS.RX_OVERFLOW	只读	0x0
6	<b>ERROR_RX_TIMEOUT</b> : 来源: SIE_STATUS.RX_TIMEOUT	只读	0x0
5	<b>ERROR_DATA_SEQ</b> : 来源: SIE_STATUS.DATA_SEQ_ERROR	只读	0x0
4	<b>BUFF_STATUS</b> : 当BUFF_STATUS中任意位被置位时触发。通过清除BUFF_STATUS的所有位来复位。	只读	0x0
3	<b>TRANS_COMPLETE</b> : 每当SIE_STATUS.TRANS_COMPLETE被置位时触发。 通过写入该位来复位。	只读	0x0
2	<b>HOST_SOF</b> : 主机——每当主机发送SOF (帧起始标志) 时触发。 通过读取 SOF_RD 清除	只读	0x0
1	<b>HOST_RESUME</b> : 主机——当设备唤醒主机时触发。通过写入 SIE_STATUS.RESUME 清除	只读	0x0
0	<b>HOST_CONN_DIS</b> : 主机——当设备连接或断开 (即SIE_STATUS.SPEED发生变化) 时触发。通过写入SIE_STATUS.SPEED来复位	只读	0x0

## USB: SOF\_TIMESTAMP\_RAW寄存器

偏移量: 0x100

表1221。  
SOF\_TIMESTAMP\_R  
AW寄存器

位	描述	类型	复位
31:21	保留。	-	-
20:0	仅限设备模式。48MHz 下自由运行物理层时钟计数器的原始值。用于计算 SOF 事件间时间。	只读	0x000000

## USB: SOF\_TIMESTAMP\_LAST寄存器

偏移量: 0x104

表1222。  
SOF\_TIMESTAMP\_LA  
ST寄存器

位	描述	类型	复位
31:21	保留。	-	-
20:0	仅限设备模式。上次 SOF 事件发生时 48MHz 自由运行物理层时钟计数器的值 ◦	只读	0x000000

## USB: SM\_STATE寄存器

偏移量: 0x108

表1223。  
SM\_STATE寄存器

位	描述	类型	复位
31:12	保留。	-	-
11:8	<b>RX_DASM</b>	只读	0x0
7:5	<b>BC_STATE</b>	只读	0x0
4:0	<b>状态</b>	只读	0x00

## USB: EP\_TX\_ERROR寄存器

偏移: 0x10c

### 描述

每端点的发送错误计数。向各字段写入以将计数器清零。

表1224。  
EP\_TX\_ERROR  
R寄存器

位	描述	类型	复位
31:30	<b>EP15</b>	WC	0x0
29:28	<b>EP14</b>	WC	0x0
27:26	<b>EP13</b>	WC	0x0
25:24	<b>EP12</b>	WC	0x0
23:22	<b>EP11</b>	WC	0x0
21:20	<b>EP10</b>	WC	0x0
19:18	<b>EP9</b>	WC	0x0
17:16	<b>EP8</b>	WC	0x0
15:14	<b>EP7</b>	WC	0x0
13:12	<b>EP6</b>	WC	0x0
11:10	<b>EP5</b>	WC	0x0
9:8	<b>EP4</b>	WC	0x0
7:6	<b>EP3</b>	WC	0x0

位	描述	类型	复位
5:4	<b>EP2</b>	WC	0x0
3:2	<b>EP1</b>	WC	0x0
1:0	<b>EP0</b>	WC	0x0

## USB: EP\_RX\_ERROR 寄存器

偏移量: 0x110

### 描述

每端点的接收错误计数。向各字段写入以将计数器清零。

表 1225。  
EP\_RX\_ERROR  
寄存器

位	描述	类型	复位
31	<b>EP15_SEQ</b>	WC	0x0
30	<b>EP15_TRANSACTION</b>	WC	0x0
29	<b>EP14_SEQ</b>	WC	0x0
28	<b>EP14_TRANSACTION</b>	WC	0x0
27	<b>EP13_SEQ</b>	WC	0x0
26	<b>EP13_TRANSACTION</b>	WC	0x0
25	<b>EP12_SEQ</b>	WC	0x0
24	<b>EP12_TRANSACTION</b>	WC	0x0
23	<b>EP11_SEQ</b>	WC	0x0
22	<b>EP11_TRANSACTION</b>	WC	0x0
21	<b>EP10_SEQ</b>	WC	0x0
20	<b>EP10_TRANSACTION</b>	WC	0x0
19	<b>EP9_SEQ</b>	WC	0x0
18	<b>EP9_TRANSACTION</b>	WC	0x0
17	<b>EP8_SEQ</b>	WC	0x0
16	<b>EP8_TRANSACTION</b>	WC	0x0
15	<b>EP7_SEQ</b>	WC	0x0
14	<b>EP7_TRANSACTION</b>	WC	0x0
13	<b>EP6_SEQ</b>	WC	0x0
12	<b>EP6_TRANSACTION</b>	WC	0x0
11	<b>EP5_SEQ</b>	WC	0x0
10	<b>EP5_TRANSACTION</b>	WC	0x0
9	<b>EP4_SEQ</b>	WC	0x0
8	<b>EP4_TRANSACTION</b>	WC	0x0
7	<b>EP3_SEQ</b>	WC	0x0
6	<b>EP3_TRANSACTION</b>	WC	0x0
5	<b>EP2_SEQ</b>	WC	0x0

位	描述	类型	复位
4	<b>EP2_TRANSACTION</b>	WC	0x0
3	<b>EP1_SEQ</b>	WC	0x0
2	<b>EP1_TRANSACTION</b>	WC	0x0
1	<b>EP0_SEQ</b>	WC	0x0
0	<b>EP0_TRANSACTION</b>	WC	0x0

## USB: DEV\_SM\_WATCHDOG 寄存器

偏移: 0x114

### 描述

看门狗：若设备长时间处于非空闲状态，强制状态机置于空闲并触发中断。计数器在每次状态转换时复位。

先在 enable 低电平时设置限制值，再设使能。

表 1226.  
DEV\_SM\_WATCHDOG  
寄存器

位	描述	类型	复位
31:21	保留。	-	-
20	<b>触发</b>	WC	0x0
19	重置：设置为 1 以在看门狗超时后强制复位设备状态机	读写	0x0
18	<b>启用</b>	读写	0x0
17:0	<b>限制</b>	读写	0x00000

## 12.8. 系统定时器

### 12.8.1. 概述

RP2350 的系统定时器外设为系统提供微秒时间基准，并基于该时间基准产生中断。RP2350 具有两个系统定时器实例：**TIMER0** 和 **TIMER1**。这允许两个定时器分别独立控制，且各位于不同的安全域中。支持以下功能：

- 单个 64 位计数器，每微秒递增一次
  - 通过一对锁存寄存器读取，实现 32 位总线上的无竞争读取
- 四个报警器，匹配计数器低 32 位时生成中断请求

该定时器使用由滴答发生器（参见第 8.5 节）产生的单微秒参考信号，参考信号源自参考时钟（图 33），通常直接连接至晶体振荡器（第 8.2 节）。

64 位计数器实际上无法溢出（在 1 MHz 频率下可持续数千年），因此系统定时器在实际应用中具备完全的单调性。

#### 12.8.1.1. RP2040 的变更

- RP2350 现有两个定时器实例：**TIMER0** 和 **TIMER1**
- 在 RP2350 中，每个定时器的节拍源来自系统级节拍发生器（见第 8.5 节）
- RP2350 新增两个寄存器：**LOCKED** 用于禁止写入定时器，**SOURCE** 允许定时器

计数系统时钟周期，而非 $1\mu\text{s}$ 节拍

### 12.8.1.2. RP2350上的其他定时器资源

系统定时器为软件提供全局时间基准。RP2350具备多种其他可编程计数资源，可实现定期中断或触发DMA传输。

- PWM（第12.5节）包含12个16位可编程计数器。这些计数器：
  - 以最高系统速度运行
  - 能够向两个系统IRQ线路中的任意一条生成中断
  - 可通过DMA持续重新编程
  - 能够触发对其他外设的DMA传输
- 12个PIO状态机（第11章）能够以系统速度计数32位值，并生成中断。
- DMA（第12.6节）设有四个内部节奏计时器，能够在固定时间间隔触发传输。
- 每个Cortex-M33内核（第3.7节）均配备标准的24位SysTick计时器，可计数微秒时钟滴答（第8.5节）或系统时钟。
- SIO配备标准的64位RISC-V平台计时器（第3.1.8节），Arm和RISC-V软件均可使用该计时器。
- 电源管理器（第6章）包含64位计时器（AON Timer），名义上计数毫秒（参见第12.10节）。当芯片处于最低功耗状态且所有可切换电源域均关闭时，该计时器为唯一运行计时器。该功能用于计划开机时间。

### 12.8.2. 计数器

计时器配备64位计数器，但RP2350仅具备32位数据总线。这意味着 `TIME` 值通过一对寄存器访问。该寄存器对如下：

- `TIMEHW` 和 `TIMELW` 用于写入时间
- `TIMEHR` 和 `TIMELR` 用于读取时间

使用这些寄存器对时，应先访问低位寄存器 `L`，随后访问高位寄存器 `H`。在读取时，读取 `L` 寄存器会锁存 `H` 寄存器中的值，从而保证时间的准确性。若需读取未锁存的原始时间，请使用 `TIMERAWH` 与 `TIMERAWL`。

#### 警告

如其他软件可能在使用计时器，请勿向 `TIMEHW` 和 `TIMELW` 寄存器写入，避免强制设置新的时间值。SDK利用时间值进行超时、计时等功能，要求该值单调递增。

### 12.8.3. 警报

计时器配备4个警报，且每个警报均输出单独中断信号。报警通过匹配64位计数器的低32位实现，这意味着报警最多可以在未来 $2^{32}$ 微秒内触发。相当于：

- $2^{32} \div 10^6$ : 约4295秒
- $4295 \div 60$ : 约72分钟

**① 注意**

该计时器支持从一微妙到一小时的报警间隔。如需更长时间的报警，请参见第12.10节。

启用报警步骤如下：

1. 通过向INT0寄存器中相应的报警位写入值（例如`(1 << 0)`对应 ALARM0）以启用计时器中断。
2. 在处理器端启用相应的计时器中断（详见第3.2节）。
3. 向 ALARM0 写入期望的中断触发时间（即当前 TIMERAWL 值加上期望的报警时间，单位为微妙）。向 ALARM 寄存器写入时间后，作为副作用会设置 ARMED位。

当报警触发后，ARMED位将清零为 0。要清除锁存中断，请向 INTR寄存器中的相应位写入 1。

## 12.8.4. 程序员模型

**① 注意**

计时器的节拍（参见第8.5节）必须在运行中，计时器才能开始计数。SDK在平台初始化代码中启动此节拍。

### 12.8.4.1 读取时间

**① 注意**

此处的时间指自计时器启动以来经过的微妙数，而非时钟时间。有关时钟功能，请参见第12.10节。

要读取64位时间，先读取 TIMELR，再读取 TIMEHR。读取 TIMELR会锁存（停止） TIMEHR的值，直到读取 TIMEHR为止。由于RP2350具有两个核心，若第二核心执行的代码也访问计时器，或计时器在中断处理程序与线程模式下并发读取，则此操作不安全。如果一个核心读取 TIMELR，紧接着另一个核心读取 TIMEHR， TIMEHR中的值不一定准确。以下示例展示了获取64位时间的最简形式：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/timer/timer\\_lowlevel/timer\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/timer/timer_lowlevel/timer_lowlevel.c) 第15至23行

```

15 // 从定时器获得64位时间的最简形式。
16 // 由于锁存机制，双核心调用时不安全
17 // 因此SDK中未采用此实现方式
18 static uint64_t get_time(void) {
19 // 读取低位会锁存高位值
20 uint32_t lo = timer_hw->timelr;
21 uint32_t hi = timer_hw->timehr;
22 return ((uint64_t) hi << 32u) | lo;
23 }
```

该SDK提供了一个 time\_us\_64函数，采用更为彻底的方法获取64位时间，利用了 TIMERAWH和 TIMERAWL寄存器。由于 RAW寄存器不具备锁存功能， time\_us\_64函数可以安全地被多个核心同时调用。

SDK链接: [https://github.com/raspberrypi/pico-sdk/blob/master/src/p2\\_common/hardware\\_timer/timer.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/p2_common/hardware_timer/timer.c) 第57至73行

```

57 uint64_t timer_time_us_64(timer_hw_t *timer) {
58 // 需要确保计时器的高32位
59 // 不发生变化，因此先读取高位
60 uint32_t hi = timer->timerawh;
61 uint32_t lo;
62 do {
63 // 读取低32位
64 lo = timer->timerawl;
65 // 现在再次读取高32位,
66 // 并检查其是否未递增。如若递增，则循环重新读取
67 // 再读一次低32位，以获得准确数值
68 uint32_t next_hi = timer->timerawh;
69 if (hi == next_hi) break;
70 hi = next_hi;
71 } while (true);
72 return ((uint64_t) hi << 32u) | lo;
73 }
```

#### 12.8.4.2. 设置闹钟

独立定时器示例`timer_lowlevel`演示了如何在硬件层面设置闹钟，而不使用SDK提供的定时器抽象。要使用这些抽象，请参见第12.8.4.4节。

Pico示例: [https://github.com/raspberrypi/pico-examples/blob/master/timer/timer\\_lowlevel/timer\\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/timer/timer_lowlevel/timer_lowlevel.c) 第27至74行

```

27 // 使用闹钟0
28 #define ALARM_NUM 0
29 #define ALARM_IRQ timer_hardware_alarm_get_irq_num(timer_hw, ALARM_NUM)
30
31 // 闹钟中断处理函数
32 static volatile bool alarm_fired;
33
34 static void alarm_irq(void) {
35 // 清除闹钟中断请求
36 hw_clear_bits(&timer_hw->intr, 1u << ALARM_NUM);
37
38 // 假设闹钟0已触发
39 printf("闹钟中断请求已触发\n");
40 alarm_fired = true;
41 }
42
43 static void alarm_in_us(uint32_t delay_us) {
44 // 使能我们的闹钟中断（定时器会输出4个闹钟中断）
45 hw_set_bits(&timer_hw->inte, 1u << ALARM_NUM);
46 // 设置闹钟中断对应的中断处理函数
47 irq_set_exclusive_handler(ALARM_IRQ, alarm_irq);
48 // 使能闹钟中断
49 irq_set_enabled(ALARM_IRQ, true);
50 // 在模块和处理器级别使能中断
51
52 // 闹钟计数器仅为32位，若延迟时间超过此值
53 // 需谨慎处理并记录高位部分
54 // 位
55 uint64_t target = timer_hw->timerawl + delay_us;
56
57 // 将目标时间的低32位写入闹钟寄存器，
58 // 以激活闹钟
59 timer_hw->alarm[ALARM_NUM] = (uint32_t) target;
```

```

60 }
61
62 int main() {
63 stdio_init_all();
64 printf("Timer lowlevel!\n");
65
66 // 每2秒设置一次闹钟
67 while (1) {
68 alarm_fired = false;
69 alarm_in_us(1000000 * 2);
70 // 等待闹钟触发
71 while (!alarm_fired);
72 }
73 }
```

#### 12.8.4.3. 忙等待

如果不使用闹钟等待某段时间，可改用while循环。SDK 提供了多种 `busy_wait_` 函数以实现此功能：

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_timer/timer.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_timer/timer.c) 第77至122行

```

77 void timer_busy_wait_us_32(timer_hw_t *timer, uint32_t delay_us) {
78 if (0 <= (int32_t)delay_us) {
79 // 我们仅允许31位，否则下面的循环中可能会出现竞态条件
80 // 该值极其接近2^32
81 uint32_t start = timer->timerawl;
82 while (timer->timerawl - start < delay_us) {
83 tight_loop_contents();
84 }
85 } else {
86 busy_wait_us(delay_us);
87 }
88 }
89
90 void timer_busy_wait_us(timer_hw_t *timer, uint64_t delay_us) {
91 uint64_t base = timer_time_us_64(timer);
92 uint64_t target = base + delay_us;
93 if (target < base) {
94 target = (uint64_t)-1;
95 }
96 absolute_time_t t;
97 update_us_since_boot(&t, target);
98 timer_busy_wait_until(timer, t);
99 }
100
101 void timer_busy_wait_ms(timer_hw_t *timer, uint32_t delay_ms)
102 {
103 if (delay_ms <= 0xffffffff / 1000) {
104 timer_busy_wait_us_32(timer, delay_ms * 1000);
105 } else {
106 timer_busy_wait_us(timer, delay_ms * 1000ull);
107 }
108 }
109
110 void timer_busy_wait_until(timer_hw_t *timer, absolute_time_t t) {
111 uint64_t target = to_us_since_boot(t);
112 uint32_t hi_target = (uint32_t)(target >> 32u);
113 uint32_t hi = timer->timerawh;
114 while (hi < hi_target) {
```

```

115 hi = timer->timerawh;
116 tight_loop_contents();
117 }
118 while (hi == hi_target && timer->timerawl < (uint32_t) target) {
119 hi = timer->timerawh;
120 tight_loop_contents();
121 }
122 }
```

#### 12.8.4.4. 使用 SDK 的完整示例

Pico 示例：[https://github.com/raspberrypi/pico-examples/blob/master/timer/hello\\_timer/hello\\_timer.c](https://github.com/raspberrypi/pico-examples/blob/master/timer/hello_timer/hello_timer.c) 第 11-57 行

```

11 volatile bool timer_fired = false;
12
13 int64_t alarm_callback(alarm_id_t id, __unused void *user_data) {
14 printf("定时器%d触发! \n", (int) id);
15 timer_fired = true;
16 // 此处可返回以微秒为单位的值，用于将来触发
17 return 0;
18 }
19
20 bool repeating_timer_callback(__unused struct repeating_timer *t) {
21 printf("重复触发时间%lld\n", time_us_64());
22 return true;
23 }
24
25 int main() {
26 stdio_init_all();
27 printf("你好，定时器! \n");
28
29 // 2 秒后调用 alarm_callback
30 add_alarm_in_ms(2000, alarm_callback, NULL, false);
31
32 // 等待 alarm_callback 设置 timer_fired
33 while (!timer_fired) {
34 tight_loop_contents();
35 }
36
37 // 创建一个调用 repeating_timer_callback 的重复定时器。
38 // 如果延迟 > 0，则该延迟为上一次回调结束与下一次回调开始之间的间隔。
39
40 // 上一次回调调用开始后的500毫秒
41 struct repeating_timer timer;
42 add_repeating_timer_ms(500, repeating_timer_callback, NULL, &timer);
43 sleep_ms(3000);
44 bool cancelled = cancel_repeating_timer(&timer);
45 printf("cancelled... %d\n", cancelled);
46 sleep_ms(2000);
47
48 // 负延迟表示我们将调用 repeating_timer_callback，并在
49 // 500毫秒后再次调用，无论回调运行耗时多久
50 add_repeating_timer_ms(-500, repeating_timer_callback, NULL, &timer);
51 sleep_ms(3000);
52 cancelled = cancel_repeating_timer(&timer);
53 printf("cancelled... %d\n", cancelled);
54 sleep_ms(2000);
55 printf("Done\n");
```

```

56 return 0;
57 }

```

## 12.8.5. 寄存器列表

TIMER0和TIMER1寄存器的基地址分别为0x400b0000和0x400b8000（SDK中定义为TIMER0\_BASE和TIMER1\_BASE）。

表 1227.  
TIMER 寄存器列表

偏移量	名称	说明
0x00	<a href="#">TIMEHW</a>	写入 time 寄存器的第 63:32 位 必须始终先写入 timelw，然后再写入 timehw
0x04	<a href="#">TIMEHW</a>	写入 time 寄存器的第 31:0 位 写入操作在写入 timehw 之前不会反映到 time
0x08	<a href="#">TIMEHR</a>	读取 time 寄存器的第 63:32 位 必须始终先读取 timelr，然后再读取 timehr
0x0c	<a href="#">TIMELR</a>	读取 time 寄存器的第 31:0 位
0x10	<a href="#">ALARM0</a>	启动报警 0 并配置其触发时间。 报警启动后，当 TIMER_ALARM0 等于 TIMELR 时触发。 报警触发后将自动解除启动，也可通过 ARMED 状态寄存器提前解除。
0x14	<a href="#">ALARM1</a>	启动报警 1 并配置其触发时间。 一旦启动，当 TIMER_ALARM1 等于 TIMELR 时，警报触发。 报警触发后将自动解除启动，也可通过 ARMED 状态寄存器提前解除。
0x18	<a href="#">ALARM2</a>	启动警报 2，并配置其触发时间。 一旦启动，当 TIMER_ALARM2 等于 TIMELR 时，警报触发。警报触发后自动解除，也可通过 ARMED 状态寄存器提前解除。
0x1c	<a href="#">ALARM3</a>	启动警报 3，并配置其触发时间。 一旦启动，当 TIMER_ALARM3 等于 TIMELR 时，警报触发。 报警触发后将自动解除启动，也可通过 ARMED 状态寄存器提前解除。
0x20	<a href="#">ARMED</a>	指示每个警报的启动或解除状态。 写入对应的 ALARMx 寄存器以启动警报。 警报触发后自动解除，但写入 1 可立即解除，无需等待触发。
0x24	<a href="#">TIMERAWH</a>	直接读取时间的第 63 至 32 位（无副作用）
0x28	<a href="#">TIMERAWL</a>	直接读取时间的第 31 至 0 位（无副作用）
0x2c	<a href="#">DBGPAUSE</a>	将位设置为高电平以启用对应调试端口激活时的暂停功能
0x30	<a href="#">暂停</a>	置高电平以暂停计时器
0x34	<a href="#">已锁定</a>	设置锁定位以禁用对计时器的写访问 设置后不可清除（除非复位）

偏移量	名称	说明
0x38	来源	选择计时器的时钟源。默认使用ticks模块中配置的正常时钟（通常配置为1微秒）。写入1时将忽略ticks，改为计数clk_sys周期。
0x3c	中断	原始中断
0x40	INTE	中断使能
0x44	INTF	中断触发
0x48	INTS	掩码与强制后的中断状态

## 计时器：TIMEHW寄存器

偏移: 0x00

表1228. TIMEHW  
寄存器

位	描述	类型	复位
31:0	写入 time 寄存器的第 63:32 位 必须始终先写入 timelw，然后再写入 timehw	WF	0x00000000

## 计时器：TIMELW寄存器

偏移: 0x04

表1229. TIMELW  
寄存器

位	描述	类型	复位
31:0	写入 time 寄存器的第 31:0 位 写入操作在写入 timehw 之前不会反映到 time	WF	0x00000000

## 计时器：TIMEHR寄存器

偏移: 0x08

表1230. TIMEHR  
寄存器

位	描述	类型	复位
31:0	读取 time 寄存器的第 63:32 位 必须始终先读取 timelr，然后再读取 timehr	只读	0x00000000

## 计时器：TIMELR寄存器

偏移: 0x0c

表1231. TIMELR  
寄存器

位	描述	类型	复位
31:0	读取 time 寄存器的第 31:0 位	只读	0x00000000

## 计时器：ALARM0寄存器

偏移: 0x10

表1232. ALARM0  
寄存器

位	描述	类型	复位
31:0	启动报警 0 并配置其触发时间。 报警启动后，当 TIMER_ALARM0 等于 TIMELR 时触发。 报警触发后将自动解除启动，也可通过 ARMED 状态寄存器提前解除。	读写	0x00000000

## 定时器：ALARM1 寄存器

偏移：0x14

表 1233. ALARM1  
寄存器

位	描述	类型	复位
31:0	启动报警 1 并配置其触发时间。 一旦启动，当 TIMER_ALARM1 等于 TIMELR 时，警报触发。 报警触发后将自动解除启动，也可通过 ARMED 状态寄存器提前解除。	读写	0x00000000

## 定时器：ALARM2 寄存器

偏移：0x18

表 1234. ALARM2  
寄存器

位	描述	类型	复位
31:0	启动警报 2，并配置其触发时间。 一旦启动，当 TIMER_ALARM2 等于 TIMELR 时，警报触发。警 报触发后自动解除，也可通过 ARMED 状态寄存 器提前解除。	读写	0x00000000

## 定时器：ALARM3 寄存器

偏移：0x1c

表 1235. ALARM3  
寄存器

位	描述	类型	复位
31:0	启动警报 3，并配置其触发时间。 一旦启动，当 TIMER_ALARM3 等于 TIMELR 时，警报触发。 警报触发后将自动解除启动，也可通过 ARMED 状态寄存器提前解除。	读写	0x00000000

## 定时器：ARMED 寄存器

偏移：0x20

表 1236. ARMED  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:0	指示每个警报的启动或解除状态。 写入对应的 ALARMx 寄存器以启动警报。 警报触发后自动解除，但写入 1 可立即解除，无需等待触发。	WC	0x0

## 定时器：TIMERAWH 寄存器

偏移：0x24

表 1237。  
TIMERAWH 寄存器

位	描述	类型	复位
31:0	直接读取时间的第 63 至 32 位（无副作用）	只读	0x00000000

## 定时器：TIMERAWL 寄存器

偏移量: 0x28

表 1238。  
TIMERAWL 寄存器

位	描述	类型	复位
31:0	直接读取时间的第 31 至 0 位（无副作用）	只读	0x00000000

## 定时器：DBGPAUSE 寄存器

偏移量: 0x2c

### 说明

将位设为高电平以在对应调试端口激活时启用暂停

表 1239。  
DBGPAUSE 寄存器

位	描述	类型	复位
31:3	保留。	-	-
2	<b>DBG1:</b> 当处理器1处于调试模式时暂停	读写	0x1
1	<b>DBG0:</b> 当处理器0处于调试模式时暂停	读写	0x1
0	保留。	-	-

## TIMER：PAUSE 寄存器

偏移量: 0x30

表 1240. PAUSE  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	置高电平以暂停计时器	读写	0x0

## TIMER：LOCKED 寄存器

偏移量: 0x34

表 1241. LOCKED  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	设置锁定位以禁用对计时器的写访问 设置后不可清除（除非复位）	读写	0x0

## TIMER：SOURCE 寄存器

偏移: 0x38

### 描述

选择计时器的时钟源。默认使用ticks模块中配置的正常时钟（通常配置为1微秒）。写入1时将忽略ticks，改为计数clk\_sys周期。

表 1242. SOURCE  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>CLK_SYS</b>	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → TICK		
	0x1 → CLK_SYS		

## TIMER: INTR 寄存器

偏移量: 0x3c

### 描述

原始中断

表 1243. INTR 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>ALARM_3</b>	WC	0x0
2	<b>ALARM_2</b>	WC	0x0
1	<b>ALARM_1</b>	WC	0x0
0	<b>ALARM_0</b>	WC	0x0

## TIMER: INTF 寄存器

偏移量: 0x44

### 描述

中断强制

表 1244. INTF 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>ALARM_3</b>	读写	0x0
2	<b>ALARM_2</b>	读写	0x0
1	<b>ALARM_1</b>	读写	0x0
0	<b>ALARM_0</b>	读写	0x0

## TIMER: INTS 寄存器

偏移量: 0x44

### 描述

中断强制

表 1245. INTS 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>ALARM_3</b>	读写	0x0
2	<b>ALARM_2</b>	读写	0x0
1	<b>ALARM_1</b>	读写	0x0
0	<b>ALARM_0</b>	读写	0x0

## TIMER: INTS 寄存器

偏移: 0x48

**说明**

掩码与强制后的中断状态

表1246。INTS  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>ALARM_3</b>	只读	0x0
2	<b>ALARM_2</b>	只读	0x0
1	<b>ALARM_1</b>	只读	0x0
0	<b>ALARM_0</b>	只读	0x0

## 12.9. 看门狗

### 12.9.1. 概述

看门狗是一种倒计时定时器，当计数归零时可配置为重置选定组件。在正常运行时，它会周期性地载入一个非零值以防止复位发生。如果芯片锁死或软件陷入循环，复位功能将允许系统恢复。

看门狗可被任何芯片级复位（参见第7.3节）重置。芯片级复位的来源如下：

- 上电复位 (POR)
- 欠压检测 (BOD)
- 外部复位 (来自RUN引脚)
- 调试器复位请求
- 救援调试端口请求
- 看门狗——由看门狗触发的芯片级复位将重置看门狗
- SWCORE掉电
- 瞬态故障检测器
- 调试器HWD复位请求

详见第7.3.3节。

### 12.9.2. 与RP2040的差异

在RP2040中，看门狗内含一个用于生成1μs计时的滴答发生器。该信号也分发至系统定时器。在RP2350中，看门狗改为接受来自系统级滴答模块的滴答输入。

请参阅第8.5节。

在RP2040上，看门狗可以触发PSM（上电状态机）序列以重置系统组件，或用于重置选定的子系统组件。在RP2350上，看门狗还可触发芯片级复位。

### 12.9.3. 看门狗计数器

看门狗计数器通过LOAD寄存器加载。当前值可在CTRL.TIME寄存器中查看。

## 12.9.4. 控制看门狗复位级别

要控制由看门狗事件触发的复位级别，请使用看门狗寄存器块外的寄存器：

- `POWMAN_WATCHDOG` 允许看门狗触发芯片级复位。
- `PSM_WDSEL` 允许看门狗通过执行完整或部分PSM序列（上电状态机）触发系统复位。
- `RESETS_WDSEL` 允许看门狗触发子系统复位。

详见复位章节，第7章。

## 12.9.5. 暂存寄存器

看门狗包含八个32位临时寄存器，可在芯片软复位期间保存信息。临时寄存器将在以下情况下复位：

- 看门狗用于触发芯片级复位。
- a `rst_n_run` 事件发生，触发条件为切换RUN引脚或循环数字核心电源（DVDD）

启动只读存储器在引导时检查看门狗擦写寄存器中的魔术数字。您可以利用此功能将芯片软复位至用户指定的代码。更多信息详见第5.2.4节。

### 注意

POWMAN SCRATCH0至SCRATCH7提供额外的通用擦写寄存器。这些寄存器在切换核心域电源时数值依然保持。

## 12.9.6. 程序员模型

SDK提供硬件看门狗驱动程序以控制看门狗。

### 12.9.6.1. 启用看门狗

SDK地址：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_watchdog/watchdog.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_watchdog/watchdog.c) 行47至76

```

47 // 此辅助函数由watchdog_enable和watchdog_reboot共同调用
48 void _watchdog_enable(uint32_t delay_ms, bool pause_on_debug) {
49 valid_params_if(HARDWARE_WATCHDOG, delay_ms <= WATCHDOG_LOAD_BITS / (1000 *
50 WATCHDOG_XFACTOR));
51
52 // 重置除ROSC和XOSC之外的所有模块
53 hw_set_bits(&psm_hw->wdsel, PSM_WDSEL_BITS & ~(PSM_WDSEL_ROSC_BITS |
54 PSM_WDSEL_XOSC_BITS));
55
55 uint32_t dbg_bits = WATCHDOG_CTRL_PAUSE_DBG0_BITS |
56 WATCHDOG_CTRL_PAUSE_DBG1_BITS |
57 WATCHDOG_CTRL_PAUSE_JTAG_BITS;
58
59 if (pause_on_debug) {
60 hw_set_bits(&watchdog_hw->ctrl, dbg_bits);
61 } else {
62 hw_clear_bits(&watchdog_hw->ctrl, dbg_bits);
63 }
64 }
```

```

65 if (!delay_ms) {
66 hw_set_bits(&watchdog_hw->ctrl, WATCHDOG_CTRL_TRIGGER_BITS);
67 } else {
68 load_value = delay_ms * (1000 * WATCHDOG_XFACTOR);
69 if (load_value > WATCHDOG_LOAD_BITS)
70 load_value = WATCHDOG_LOAD_BITS;
71
72 watchdog_update();
73
74 hw_set_bits(&watchdog_hw->ctrl, WATCHDOG_CTRL_ENABLE_BITS);
75 }
76 }
```

### 12.9.6.2. 更新看门狗计数器

SDK: [https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\\_common/hardware\\_watchdog/watchdog.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_watchdog/watchdog.c) 第24至28行

```

24 static uint32_t load_value;
25
26 void watchdog_update(void) {
27 watchdog_hw->load = load_value;
28 }
```

### 12.9.6.3. 使用说明

Pico 示例仓库提供了一个hello\_watchdogexample，使用hardware\_watchdog演示看门狗的功能。

Pico 示例: [https://github.com/raspberrypi/pico-examples/blob/master/watchdog/hello\\_watchdog/hello\\_watchdog.c](https://github.com/raspberrypi/pico-examples/blob/master/watchdog/hello_watchdog/hello_watchdog.c) 第11至33行

```

11 int main() {
12 stdio_init_all();
13
14 if (watchdog_enable_caused_reboot()) {
15 printf("由看门狗重启! \n");
16 return 0;
17 } else {
18 printf("正常启动\n");
19 }
20
21 // 启用看门狗，必须每100ms更新一次看门狗，否则芯片将会重启
22 // 第二个参数表示调试时暂停，即单步调试时看门狗将暂停
23 watchdog_enable(100, 1);
24
25 for (uint i = 0; i < 5; i++) {
26 printf("正在更新看门狗 %d\n", i);
27 watchdog_update();
28 }
29
30 // 进入无限循环不更新看门狗，导致设备重启
31 printf("等待被看门狗重启\n");
32 while(1);
33 }
```

## 12.9.7. 寄存器列表

看门狗寄存器的基地址为 **0x400d8000** (在SDK中定义为 WATCHDOG\_BASE)。

表1247。看门狗寄存器列表

偏移量	名称	说明
0x00	CTRL	看门狗控制 rst_wdsel寄存器决定当看门狗触发时哪些子系统将被复位。 看门狗可通过软件触发。
0x04	LOAD	加载看门狗定时器。最大设置值为0xffffffff，约等于16秒。
0x08	REASON	记录上次复位的原因。在硬件复位情况下，两位均为零。  此外，自RP2350起，任一核心的调试器热复位（SYSRESETREQ或hartreset）亦会清除看门狗原因寄存器，以致调试器下加载的软件在看门狗超时后不会持续检测到超时状态。
0x0c	SCRATCH0	临时寄存器。信息在芯片软复位后依然保持。
0x10	SCRATCH1	临时寄存器。信息在芯片软复位后依然保持。
0x14	SCRATCH2	临时寄存器。信息在芯片软复位后依然保持。
0x18	SCRATCH3	临时寄存器。信息在芯片软复位后依然保持。
0x1c	SCRATCH4	临时寄存器。信息在芯片软复位后依然保持。
0x20	SCRATCH5	临时寄存器。信息在芯片软复位后依然保持。
0x24	SCRATCH6	临时寄存器。信息在芯片软复位后依然保持。
0x28	SCRATCH7	临时寄存器。信息在芯片软复位后依然保持。

### WATCHDOG: CTRL寄存器

偏移: 0x00

#### 描述

看门狗控制  
rst\_wdsel寄存器决定当看门狗触发时哪些子系统将被复位。  
看门狗可通过软件触发。

表1248。CTRL 寄存器

位	描述	类型	复位
31	<b>TRIGGER:</b> 触发看门狗复位	SC	0x0
30	<b>ENABLE:</b> 未启用时，看门狗定时器暂停运行	读写	0x0
29:27	保留。	-	-
26	<b>PAUSE_DBG1:</b> 当处理器1处于调试模式时暂停看门狗定时器	读写	0x1

位	描述	类型	复位
25	<b>PAUSE_DBG0</b> : 当处理器0处于调试模式时暂停看门狗定时器	读写	0x1
24	<b>PAUSE_JTAG</b> : 当JTAG访问总线时暂停看门狗定时器 结构	读写	0x1
23:0	<b>TIME</b> : 指示触发看门狗复位前的微秒数	只读	0x000000

## WATCHDOG: LOAD寄存器

偏移: 0x04

表1249. LOAD  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	加载看门狗定时器。最大设置值为0xffffffff, 约等于16秒。	WF	0x000000

## WATCHDOG: REASON寄存器

偏移: 0x08

### 说明

记录上次复位的原因。在硬件复位情况下，两位均为零。

此外，自RP2350起，任一核心的调试器热复位（SYSRESETREQ或hartreset）亦会清除看门狗原因寄存器，以致调试器下加载的软件在看门狗超时后不会持续检测到超时状态。

表1250. REASON  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>强制</b>	只读	0x0
0	<b>定时器</b>	只读	0x0

## WATCHDOG: SCRATCH0、SCRATCH1、...、SCRATCH6、SCRATCH7寄存器

偏移量: 0x0c, 0x10, ..., 0x24, 0x28

表1251.  
SCRATCH0、  
SCRATCH1、...、  
SCRATCH6、  
SCRATCH7寄存器

位	描述	类型	复位
31:0	临时寄存器。信息在芯片软复位后仍然保持。	读写	0x00000000

## 12.10. 常时开启定时器

### 12.10.1. 概述

始终开启定时器（AON定时器）是唯一在所有电源模式下持续运行的定时器。它可用作实时计数器或间隔定时器，并内置报警器，可用于触发加电事件或中断。它集成了一个64位计数器，设计用于计数1ms滴答，但滴答发生器可根据需要配置为更快或更慢运行。请注意，AON定时器的滴答发生器独立于芯片上所有其他滴答发生器。

默认滴答源为32kHz片上低功耗振荡器（LPOSOC），详见第8.4节。LPOSOC频率不精确，且可能因电压和温度变化而波动。当芯片核心供电时，滴答源可切换至片上晶体振荡器（XOSC），以获得更高精度。如果在芯片核心

断电时也需更高精度，则可由外部来源提供32kHz时钟或1ms滴答。或者，AON定时器可与外部1Hz信号同步。

AON定时器集成于电源管理器（POWMAN）中，且共享POWMAN寄存器块。写入操作限制为16位，因顶部16位须包含密钥（[0x5afe](#)），以防止错误写入导致芯片锁死。与其他POWMAN寄存器不同，大多数AON定时器寄存器允许非安全软件写入。

但选择外部时钟、外部计数源及使能报警开机的寄存器，仅允许安全软件写入。

### 12.10.2. 与RP2040的差异

RP2040的实时时钟（RTC）未在RP2350中使用。RP2350在始终开启的电源域内设有定时器，用于调度开机事件，也可用作实时间计数器。AON定时器的工作机制不同于RP2040的RTC。它以64位计数毫秒数，必要时该值可用于软件中计算日期和时间。

### 12.10.3. 访问AON定时器

要启动和停止AON定时器，请写入TIMER.RUN。

要读取当前的64位AON定时器值，请使用以下两个32位只读寄存器：

- [READ\\_TIME\\_UPPER](#)
- [READ\\_TIME\\_LOWER](#)

由于AON定时器在读取时可能递增，请按照以下步骤防止读取错误：

1. 读取READ\_TIME\_UPPER
2. 读取READ\_TIME\_LOWER
3. 再次读取READ\_TIME\_UPPER
4. 若步骤1与步骤3中READ\_TIME\_UPPER的值不同，请重复上述过程。

作为实时时钟使用时，64位时间值通过4个16位寄存器设置。只有在通过写入0到TIMER.RUN停止AON定时器后，这些寄存器才能写入：

- [SET\\_TIME\\_63TO48](#)
- [SET\\_TIME\\_47TO32](#)
- [SET\\_TIME\\_31TO16](#)
- [SET\\_TIME\\_15TO0](#)

这些寄存器不用于读取时间值。

作为间隔定时器使用时，写入1到TIMER.CLEAR以清除定时器值。执行此操作时，无需停止AON定时器。TIMER.CLEAR寄存器为自清零寄存器：操作完成后其值自动恢复为0。该机制使得能够轻松实现周期性唤醒芯片或产生中断的报警功能。

### 12.10.4. 使用警报

设置报警时间时，请使用以下四个16位寄存器：

- [ALARM\\_TIME\\_63TO48](#)
- [ALARM\\_TIME\\_47TO32](#)

- [ALARM\\_TIME\\_31TO16](#)
- [ALARM\\_TIME\\_15TO0](#)

为避免误报警，请在设置报警时间前禁用报警功能。

启用报警功能，请设置 `TIMER.ALARM_ENAB`。

报警触发时，AON定时器会将报警状态标志 `TIMER.ALARM` 置位。

清除报警状态标志，请向该标志写入 1。

若要配置报警触发开机，请设置 `TIMER.PWRUP_ON_ALARM`。该功能对非安全代码不可用。

报警可配置为触发中断。中断通过以下寄存器字段以标准方式处理：

- [INTR.TIMER — 原始中断](#)
- [INTE.TIMER — 中断使能](#)
- [INTF.TIMER — 强制中断](#)
- [INTS.TIMER — 中断状态](#)

### 12.10.5. 选择AON定时器的计时源

AON定时器使用只读标志指示当前配置状态。表1252列出了1kHz AON定时器滴答支持的信号源。

表1252。AON  
定时器计时发生器

计时源	只读标志
LPOSC时钟分频	<code>TIMER.USING_LPOSC</code>
XOSC时钟分频	<code>TIMER.USING_XOSC</code>
外部1kHz计时信号	<code>TIMER.USING_GPIO_1KHZ</code>

**i 注意**

LPOSC时钟可由外部32kHz时钟替代。

#### 12.10.5.1 使用LPOSC作为AON定时器计时源

LPOSC为默认计时源，可在所有电源模式下使用。其标称频率为32.768kHz，调节精度仅达1%。AON定时器通过一款6.1位分数分频器（初值为32.768）从LPOSC导出1ms计时，分频值可调整以提升精度。由于LPOSC频率受供电电压和温度影响，除非这些条件稳定，否则其精度受限。若要修改除数，请向以下寄存器写入数据：

- [LPOSC\\_FREQ\\_KHZ\\_INT](#)（默认值：32）
- [LPOSC\\_FREQ\\_KHZ\\_FRAC](#)（默认值：0.768）

仅当 `TIMER.RUN = 0` 或 `TIMER.USING_LPOSC = 0` 时，方可向这些寄存器写入。

若时基源非 LPOSC，您可通过向 `TIMER.USE_LPOSC` 写入1切换回 LPOSC，且无需停止 AON 定时器。新选定的时基会与当前时基同步，故操作最多可能持续一个时基周期（正常运行时为1毫秒）。操作完成后，`TIMER.USE_LPOSC` 将自动清零，且 `TIMER.USING_LPOSC` 将被置位。由于采样，时间中将扣除最多两个新选时钟周期的微小误差。切换至32kHz LPO SC 时，最多扣除62微秒误差。

### 12.10.5.2 使用外部时钟替代低功耗振荡器 (LPOSC)

若LPOSC精度不足，可使用外部32.768kHz时钟。该时钟将复用至内部低功耗时钟，因此驱动所有由该时钟驱动的组件，包括电源时序器组件。外部时钟适用于所有电源模式。使用外部时钟时，可停止LPOSC（详见第8.4节）。

选择外部32kHz时钟的步骤如下：

1. 按照第12.10.7节配置GPIO源。
2. 通过设置EXT\_TIME\_REF.DRIVE\_LPCK切换至外部LPOSC。该寄存器仅在TIMER.RUN = 0且电源时序器处于非活动状态时写入。该寄存器仅允许安全代码写入。

外部32kHz时钟替代LPOSC时钟，因此AON定时器配置使用相同寄存器（详见第12.10.5.1节）：

- [TIMER.USE\\_LPOSC](#)
- [TIMER.USING\\_LPOSC](#)
- [LPOSC\\_FREQ\\_KHZ\\_INT](#)
- [LPOSC\\_FREQ\\_KHZ\\_FRAC](#)

### 12.10.5.3. 使用XOSC作为AON定时器的时钟源

XOSC时钟通过参考时钟（[clk\\_ref](#)）提供。用户必须确保参考时钟已由XOSC驱动，方可将其选作AON定时器时钟源。此为启动后的正常配置。

检查时，请确认CLK\_REF\_SELECTED = [0x4](#)。参考时钟可能是XOSC的分频版本。分频系数默认为1，可从CLK\_REF\_DIV.INT读取。若芯片使用更高速的XOSC，送入AON定时器的时钟频率不得超过29MHz。

AON定时器通过16.16位分数分频器，从XOSC导出1毫秒节拍，分频系数初始化为12000.0。此设定假设使用12MHz晶振，且参考时钟分频系数为1。如果情况并非如此，则可通过向以下寄存器写入数据来修改AON计时器的除数：

- [XOSC\\_FREQ\\_KHZ\\_INT](#)（默认值：12000）
- [XOSC\\_FREQ\\_KHZ\\_FRAC](#)（默认值：0）

仅当 TIMER.RUN = 0 或 TIMER.USING\_XOSC = 0 时，方可向这些寄存器写入数据。

要选择XOSC作为AON计时器的时钟源，请向TIMER.USE\_XOSC写入1。无需停止AON计时器即可完成此操作。新选定的时基会与当前时基同步，故操作最多可能持续一个时基周期（正常运行时为1毫秒）。操作完成后，[TIMER.USE\\_XOSC](#)会自动清零，且[TIMER.USING\\_XOSC](#)会被设置。由于采样原因，时间上会扣除新时钟周期最多两个周期的小误差。切换至12MHz XOSC时，最多会扣除167ns的误差。

当芯片核心断电时，XOSC将停止运行。如果TIMER.USING\_XOSC被设置，则断电序列器会在XOSC停止之前自动恢复至TIMER.USING\_LPOSC。

### 12.10.5.4. 使用外部1毫秒计时源

要选择外部1毫秒计时源，请按照第12.10.7节的说明配置GPIO源。然后，将1写入TIMER.USE\_GPIO\_1KHZ。无须停止AON定时器，但新选择的计时信号不会与当前计时同步，因此该操作会使时间向前推进最多1毫秒。使用外部1毫秒计时源时，建议在选择计时源后设置时间。操作完成后，TIMER.USE\_GPIO\_1KHZ将自动清零，同时TIMER.USING\_GPIO\_1KHZ将被设置。

计时信号由所选GPIO的下降沿触发。为确保正确采样，GPIO脉冲宽度和间隔均须大于LPOSC的周期（>31微秒）。此限制了外部时钟的最大频率为

16kHz。

外部1毫秒计时脉冲可在所有功率模式下使用。

### 12.10.6. 将AON定时器同步至外部1Hz时钟

在使用GPS的应用中，可能提供1秒计时脉冲。该脉冲可用于同步始终通电计时器（AON Timer），从而补偿低功耗振荡器（LPOSC）频率的不准确性。此脉冲可与任何计时脉冲源配合使用，但若所选源已经较为准确，则几乎无明显益处。

若LPOSC频率偏快，毫秒计数器将在达到1秒刻度前暂停，直至收到1秒脉冲。若LPOSC频率偏慢，1秒脉冲将使毫秒计数器加速运行，直至达到1秒刻度。此设计确保所有毫秒数计数完整，保证任何基于毫秒精度设置的警报均能准确触发。更高级的同步方法可通过软件实现。

欲启用硬件同步功能，请按照第12.10.7节配置GPIO信号源，然后向TIMER.USE\_GPIO\_1HZ寄存器写入1以启用。该操作可在任意时点执行，无需停止始终通电计时器。操作完成后，TIMER.USE\_GPIO\_1HZ寄存器自动清零，且TIMER.USING\_GPIO\_1HZ寄存器将被置位。

滴答信号由所选GPIO的下降沿触发。为确保采样正确，GPIO脉冲宽度和间隔必须大于LPOSC的周期（>31微秒）。

外部1秒滴答信号可在所有电源模式下使用。

### 12.10.7. 使用GPIO的外部时钟或滴答信号

以下功能使用GPIO作为时钟或滴答信号：

- 外部32kHz时钟源
- 外部1kHz计时信号
- 外部1Hz滴答信号

仅有4个GPIO可用于这些功能。由于它们共享相同的GPIO选择逻辑，因此只能选择其中之一。4个GPIO的集合因封装类型而异。选择由一个2位寄存器字段控制。

AON定时器使用以下GPIO：

- `EXT_TIME_REF_SOURCE_SEL = 0` → GPIO12
- `EXT_TIME_REF_SOURCE_SEL = 1` → GPIO20
- `EXT_TIME_REF_SOURCE_SEL = 2` → GPIO14
- `EXT_TIME_REF_SOURCE_SEL = 3` → GPIO22

### 12.10.8. 使用快于1毫秒的滴答信号

通过调整写入LPOSC和XOSC频率寄存器的值，可增加滴答频率。例如，若频率值除以4，则AON定时器每毫秒会滴答4次。频率寄存器允许的最小写入值为2.0，因此使用LPOSC通过此方法的最大倍增因子为16，时间分辨率达到1/16毫秒（即62.5μs）。

如前所述，外部时钟频率限制为16kHz，因此此方法的最大倍增因子同样为16。这意味着时间分辨率为1/16毫秒（62.5μs）。

上述限制可通过使用更高速的外部时钟（详见第12.10.5.2节）或保持芯片核心供电，使AON定时器持续从XOSC运行来突破。如选用更高速外部时钟，则需相应调整电源序列器的时序。

例如，假设需要1微秒的定时器精度。用户可提供外部2-25MHz时钟替代LPOSC，并以MHz单位编程LPOSC和XOSC频率寄存器，而非kHz。外部时钟最大频率为29MHz。

### 12.10.9. 寄存器列表

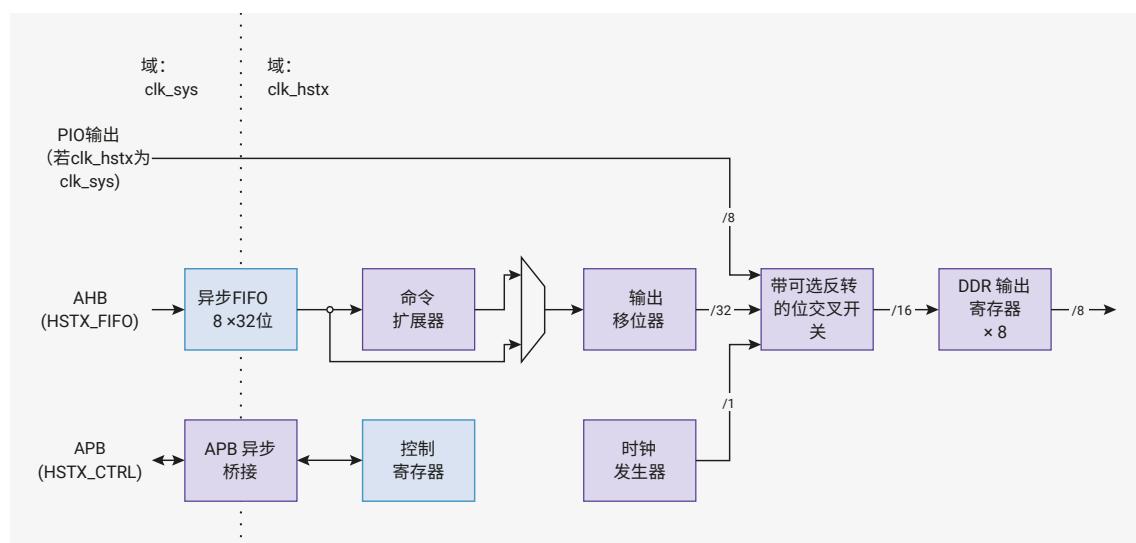
AON定时器与常电域中的电源管理子系统共享寄存器地址空间。该地址空间在本文档其他部分称为 **POWMAN**，第6.4节提供了 **POWMAN寄存器完整列表**。**AON定时器相关寄存器**如下：

- [SET\\_TIME\\_63T048](#)
- [SET\\_TIME\\_47T032](#)
- [SET\\_TIME\\_31T016](#)
- [SET\\_TIME\\_15T00](#)
- [READ\\_TIME\\_UPPER](#)
- [READ\\_TIME\\_LOWER](#)
- [ALARM\\_TIME\\_63T048](#)
- [ALARM\\_TIME\\_47T032](#)
- [ALARM\\_TIME\\_31T016](#)
- [ALARM\\_TIME\\_15T00](#)
- [定时器](#)

### 12.11. HSTX

高速串行传输（HSTX）将数据从系统时钟域传输至最多8个GPIO，传输速率独立于系统时钟。在RP2350上，GPIO12至GPIO19支持HSTX，且HSTX仅支持输出。

图126。一个32位宽的异步FIFO提供来自系统DMA的高带宽访问。  
命令扩展器操纵数据流，输出移位寄存器将在连续的HSTX时钟周期内分段传输32位数据，位交叉开关对数据位进行重排。  
输出采用双数据率：每个引脚每个时钟周期传输两位数据



HSTX通过GPIO使用DDR输出寄存器驱动数据，每个引脚每个时钟周期传输最多两位数据。HSTX将所有GPIO输出的延迟控制在300皮秒内，最大程度减少使用相邻GPIO作为伪差分驱动时的共模成分。这也有助于在时钟与输出数据同时驱动时，维持目的端的建立和保持时间。

HSTX时钟的最大频率为150 MHz，与系统时钟相同。在DDR输出操作模式下，此

每个引脚的最大数据速率为300 Mb/s。系统时钟与HSTX时钟之间的频率比无固定限制，但各时钟必须单独足够快以保证所需的吞吐量。极低的系统时钟频率配合极高的HSTX频率可能导致系统DMA带宽受限，因为DMA在每个系统时钟周期内最多允许一次HSTX FIFO写入。

### 12.11.1. 数据FIFO

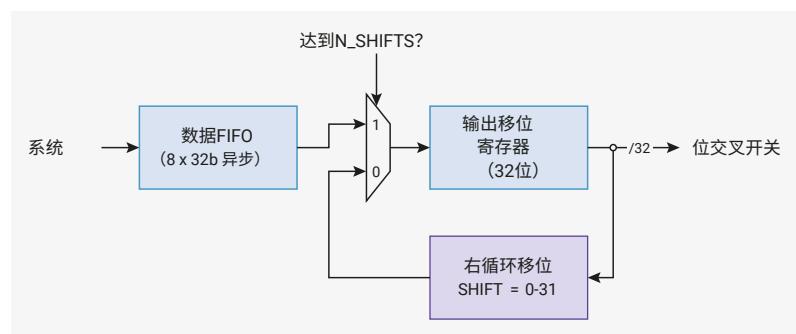
一个8级、32位宽的FIFO在系统时钟域（`clk_sys`）与HSTX时钟域（`clk_hstx`）之间缓冲数据。该FIFO通过AHB **FASTPERI**仲裁器访问，DMA可实现单周期写入。FIFO状态同样通过该总线接口提供，以支持更快速的处理器轮询IO；详见第12.11.8节。

FIFO通过与控制寄存器（第12.11.7节）不同的总线接口访问，控制寄存器因异步总线跨域访问需多个周期。该设计避免在访问FIFO时对系统DMA或 **FASTPERI**仲裁器产生总线阻塞。

HSTX端同样一次从FIFO弹出32位。FIFO中的字数据流在传递到输出移位寄存器之前，可由命令扩展器（第12.11.5节）选择性地处理。

### 12.11.2. 输出移位寄存器

图127。每个周期，输出移位寄存器要么从FIFO重新填充32位数据，要么通过右循环功能对数据进行循环。该循环功能可用于执行左移或右移位操作，以及数据重复。



HSTX的内部数据路径宽度为32位，但输出宽度较窄：每个HSTX周期输出不超过16位（8个GPIO × DDR）。输出移位寄存器用于调整这些不匹配的数据宽度。输出移位寄存器为32位宽的移位寄存器，始终一次性补充32位数据，数据来源为命令扩展器输出或直接来自数据FIFO。

输出移位寄存器的数据来源由CSR.EXPAND\_EN字段配置：

- 设置时，命令扩展器介于FIFO与输出移位寄存器之间
- 清除时，跳过命令扩展器，直接从FIFO弹出数据进入移位寄存器

每当CSR.EN为低电平时，移位寄存器将被清空。一旦配置HSTX且 EN置为高电平，移位寄存器即准备好接收数据，并将在数据可用时立即弹出。

弹出第一个数据字后，移位寄存器将在每个HSTX时钟周期执行一次移位，直至寄存器为空。移位行为由以下项配置：

- CSR.N\_SHIFTS，确定在寄存器被视为空之前应移位的次数
- CSR.SHIFT，定义每个时钟周期对移位寄存器执行的右旋转转移位

仅当CSR.EN处于低电平时，CSR.N\_SHIFTS和CSR.SHIFT方可更改。在与将EN从低电平置为高电平的寄存器写操作中同时更改这些字段是安全的。

`SHIFT × N_SHIFTS`不一定小于或等于32。例如，`SHIFT`值为31可用于每周期将寄存器左移一位；由于右旋转为模运算，在模32下，-1等价于31。

当移位寄存器即将空时，若命令扩展器或FIFO中有数据可用，将立即重新填充新数据。只要有数据可用，移位寄存器在任何周期内均不会为空。若无数据可用，

移位寄存器将变为空并停止移位，直至提供新数据。一旦提供数据，移位寄存器将重新填充并恢复移位。

### 12.11.3. 位交叉开关

位交叉开关控制输出移位寄存器的位于每个HSTX时钟周期上半周期和下半周期分别映射至相应GPIO。每个引脚均设有一个配置寄存器，位于 BIT0 至 BIT7：

- **BITx.SEL\_P** 选择移位寄存器的哪一位（0 至 31）在每个 HSTX 时钟周期的前半周期输出
- **BITx.SEL\_N** 选择移位寄存器的哪一位（0 至 31）在每个时钟周期的后半周期输出
- **BITx.INV** 对输出取反（逻辑非）
- **BITx.CLK** 指示该引脚应连接至时钟发生器（第 12.11.4 节），而非移位寄存器输出端

欲禁止 DDR 行为，请将 **SEL\_N** 设置为与 **SEL\_P** 相同。为实现差分输出，应将两个引脚配置相同，唯独 **INV** 位，一引脚设置为 1，另一引脚清零。

#### 12.11.3.1 示例：单个引脚

结合对移位寄存器的 **SHIFT** 和 **N\_SHIFTS** 控制，引脚配置决定经 HSTX 传输的数据布局。鉴于我们未必习惯于四维思考，现通过单引脚示例加以说明：

- **N\_SHIFTS = 32, SHIFT = 1, SEL\_P = 0, SEL\_N = 0:**
  - 每个HSTX时钟周期输出一位，最低有效位优先。
  - 每个周期中，移位寄存器向右移动一位，且该时刻的最低有效位在周期的两个半周期内均输出至引脚，因为 **SEL\_P** 和 **SEL\_N** 均选择相同的位。
- **N\_SHIFTS = 32, SHIFT = 31, SEL\_P = 31, SEL\_N = 31:**
  - 每个HSTX时钟周期输出一位，最高有效位优先。
  - 每个周期中，移位寄存器向左移动一位（或绕过寄存器的右端，最终处于起始位置左侧一位），此时的最高有效位输出至引脚。
- **N\_SHIFTS = 16, SHIFT = 2, SEL\_P = 0, SEL\_N = 1:**
  - 每个 HSTX 时钟周期移出两个比特，最低有效位优先。
  - 每个周期内，移位寄存器向右移动两个比特。该周期前半部分，最低有效位输出至引脚；后半部分输出相邻比特。
- **N\_SHIFTS = 16, SHIFT = 30, SEL\_P = 31, SEL\_N = 30:**
  - 每个 HSTX 时钟周期移出两个比特，最高有效位优先。
  - 每个周期内，移位寄存器向左移动两个比特。该周期前半部分，最高有效位输出至引脚；后半部分输出相邻比特。
- **N\_SHIFTS = 8, SHIFT = 4, SEL\_P = 0, SEL\_N = 0:**
  - 在8个时钟周期内，每组4位中的最低有效位依次移出。
  - 每个周期，移位寄存器向右移动四位。移位寄存器的最低有效位被输出到引脚。因此，输出到引脚的位索引分别为0、4、8、12、16、20、24和28。
- **N\_SHIFTS = 32, SHIFT = 4, SEL\_P = 0, SEL\_N = 0:**

- 与前述相同，但在刷新移位寄存器之前，将8周期模式重复四次。
- 旋转32位将恢复从FIFO或命令扩展器弹入移位寄存器的原始值。

### 12.11.3.2 示例：多引脚

移位寄存器与位交叉开关的分离允许在涉及多个引脚时，既支持压缩也支持未压缩的多位记录。例如，比较以下两种配置：

- N\_SHIFTS = 8, SHIFT = 4, BIT0.SEL\_P = 0, BIT0.SEL\_N = 2, BIT1.SEL\_P = 1, BIT1.SEL\_N = 3:**
  - 每个32位字由16对位组成，每个时钟周期向 **BIT0** 和 **BIT1** 提供两个新的位对。
  - 移位寄存器每周期移动4位，在移位寄存器最右侧的四个位引入两个新的位对。
- N\_SHIFTS = 8, SHIFT = 2, BIT0.SEL\_P = 0, BIT0.SEL\_N = 1, BIT1.SEL\_P = 16, BIT1.SEL\_N = 17:**
  - 每个32位字由一对16位值组成，每个值以每周期两位的速率移向 **BIT0** 和 **BIT1** 的对应引脚。
  - 移位寄存器每个周期前进两个位置，为位1:0的 **BIT0** 引脚引入一对新比特，同时为位17:16的 **BIT1** 引脚引入另一对新比特。

根据软件需求，可能更适合将同一周期输出的所有位打包在一起（压缩记录），或将所有经过同一引脚的位分开（非压缩记录），因此HSTX支持这两种方式。

作为最终且具体的示例，考虑TMDS（用于DVI）：每个32位字包含3个10位TMDS符号，每个符号在10个TMDS比特周期内串行传输至一对差分信号线上。为提高性能，最好利用DDR功能，使每个HSTX时钟周期等于两个TMDS比特周期。因此，可能的配置如下：

- CSR: N\_SHIFTS = 5, SHIFT = 2**
- BIT0: SEL\_P = 0, SEL\_N = 1, INV = 0**
- BIT1: SEL\_P = 0, SEL\_N = 1, INV = 1**
- BIT2: SEL\_P = 10, SEL\_N = 11, INV = 0**
- BIT3: SEL\_P = 10, SEL\_N = 11, INV = 1**
- BIT4: SEL\_P = 20, SEL\_N = 21, INV = 0**
- BIT5: SEL\_P = 20, SEL\_N = 21, INV = 1**

TMDS缺失的部分是时钟，其周期为10个TMDS位周期，或当每个引脚每周期移位两个位时为5个HSTX时钟周期。HSTX配备专用时钟发生器，因而无需将伪时钟位打包进 FIFO数据流。时钟发生器将在下一节详细介绍。

### 12.11.4. 时钟发生器

时钟发生器是一个计数器，在 **n** 个HSTX时钟周期内提供周期性信号，其配置由 **CSR.CLKDIV** 决定。时钟周期始终为1至16之间的整数个HSTX时钟周期。时钟发生器支持奇数和偶数周期，利用 **DDR** 输出以支持HSTX时钟周期中间的输出跳变。仅有单一时钟发生器——若需模拟多个时钟，应将伪时钟位打包到FIFO数据中。

时钟发生器在输出移位寄存器移位的周期内递增。通常，时钟周期应为 **CSR.N\_SHIFTS** 的因数，以确保时钟与数据维持一致的对齐。在前述TMDS示例中，设置 **CLKDIV** 为5适宜，使时钟在移位寄存器刷新时重复。此设置满足TMDS时钟周期为10位周期的要求，因每周期传输两个比特。

时钟发生器输出连接至任一其 **BITx.CLK** 位被置位的引脚（例如 **BIT0.CLK**）。要生成差分

时钟输出，请将时钟信号连接至两个引脚，并使其中一个引脚的信号反相。

CSR.CLKPHASE 字段定义时钟发生器的初始相位（计数），单位为半个 HSTX 时钟周期。当 CSR.EN 处于低电平时，时钟发生器复位并保持在该初始相位。一旦 CSR.EN 被置位且输出移位寄存器开始移位，时钟发生器开始计数。

在 CSR.EN 为低电平时，时钟发生器输出由时钟周期与初始时钟相位的关系决定：若初始时钟相位小于半个时钟周期，输出初始为低电平；否则，输出初始为高电平。时钟发生器可视为在每个生成周期的前半周期输出低电平，后半周期输出高电平。

CSR.CLKPHASE 最大值仅为 15 半个 HSTX 时钟周期。最大 CSR.CLKDIV 为 16 完整 HSTX 时钟周期：当初始相位大于或等于 180 度且时钟周期为最大时，须使用位交叉反转控制以反转时钟信号。

仅当 CSR.EN 处于低电平时，方可更改 CSR.CLKPHASE 和 CSR.CLKDIV。可在设置 EN 从低电平到高电平的同一次寄存器写入操作中安全修改它们。

#### 12.11.4.1 示例：居中对齐时钟

在传输源同步数据时，数据接收端（接收器）不得在时钟的有效沿之前过早，或在有效沿之后过早检测到数据跳变。违反这些建立时间和保持时间约束可能导致外部数据接收端的操作异常或不确定。

由于所有 HSTX 输出延迟相互平衡，您可通过将时钟跳变定位于数据跳变之间的中点，即使用居中对齐时钟方式，来满足这些时序约束。

由于时钟的位置具有半个比特时间的时间分辨率，最大数据速率为每针脚每个 HSTX 时钟周期一个比特。鉴于时钟已采用 DDR，不能通过 DDR 进一步提高数据速率。因此，对于所有 BIT0 到 BIT7，`BITx.SEL_N` 等于 `BITx.SEL_P`。

对于单数据速率数据，在上升沿有效时，请使用以下时钟发生器设置：

- `CSR.CLKDIV = 1` (1 个 HSTX 时钟周期)
- `CSR.CLKPHASE = 1` (1/2 个 HSTX 时钟周期)

时钟延迟半个 HSTX 周期，以使其与第一个数据的触发时间错开。

对于单数据速率数据，在下降沿有效时，请使用以下时钟发生器设置：

- `CSR.CLKDIV = 1` (1 个 HSTX 时钟周期)
- `CSR.CLKPHASE = 2` (1 个 HSTX 时钟周期)

或者，您可以使用与上升沿有效时钟相同的设置，通过位交叉开关配置反转时钟输出。

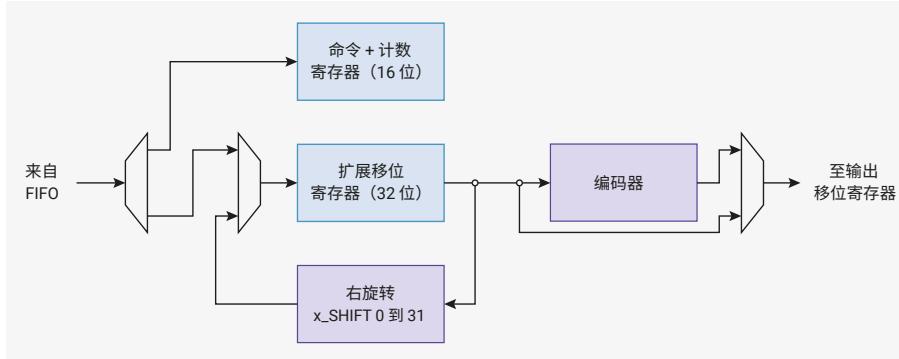
对于双数据速率数据，在上升沿和下降沿均有效时，请使用以下时钟发生器设置：

- `CSR.CLKDIV = 2` (2 个 HSTX 时钟周期)
- `CSR.CLKPHASE = 1` (1/2 个 HSTX 时钟周期)

在所有三种情况下，数据速率均相同，为每个引脚每个 HSTX 时钟周期 1 位。

#### 12.11.5. 命令扩展器

图 128。从 FIFO 弹出的是命令和数据的混合体。数据可在通过扩展移位寄存器时重复或移位，且可选择经过编码器后传递至输出移位寄存器。



命令扩展器可在线插入于数据 FIFO 与输出移位寄存器之间，以操作数据字流。一般情况下，输出流比输入流更大，因此称为扩展器。通过设置 CSR.EXPAND\_EN 启用命令扩展器。仅当 CSR.EN 为低时方可修改此字段。与同时将 EN 从低置高的寄存器写操作中修改该字段是安全的。当命令扩展器被禁用时，数据直接从数据 FIFO 传送至输出移位寄存器，且不会被扩展器修改。

当命令扩展器启用时，数据 FIFO 包含扩展器使用的混合数据与命令。每条命令由 4 位操作码和 12 位长度组成，封装于数据 FIFO 字的 16 位最低有效位中，操作码位于第 15 至 12 位，长度位于第 11 至 0 位。可用命令如下：

- 0x0: RAW
- 0x1: RAW\_REPEAT
- 0x2: TMDS
- 0x3: TMDS\_REPEAT
- 0xf: NOP

当 HSTX 首次启用且命令扩展器开启时，它期望数据 FIFO 中的第一个字为命令。若该命令非 NOP，则后续会跟随一定数量的数据，然后是下一条命令。操作以此种方式继续，数据块与命令交替出现。命令始终作为紧随其后的 FIFO 中数据的前缀。

计数字段确定该命令向下游输出移位寄存器输出的数据字数，范围为 1 至 4095。计数值 0 被保留用以表示“无限”。该命令为产生指定数量的下游数据，从数据 FIFO 读取的字数依赖于命令及 EXPAND\_SHIFT.ENC\_N\_SHIFTS 和 EXPAND\_SHIFT.RAW\_N\_SHIFTS 寄存器字段。

扩展移位寄存器在命令开始时始终从 FIFO 弹出一次。此后，带有 x\_REPEAT 后缀的命令持续通过移位寄存器循环相同内容，每当输出移位寄存器从命令扩展器读取新数据时，按 EXPAND\_SHIFT.ENC\_SHIFT 或 EXPAND\_SHIFT.RAW\_SHIFT 向右旋转。使用移位值 0 以无移位方式重复相同数据。例如，在 DVI 的水平消隐期间传输连续相同的 TMDS 控制符号时，此功能非常有用。

RP2350 仅实现了 TMDS 编码器，保留剩余的操作码空间以便将来添加更多编码器。

RAW 和 RAW\_REPEAT 命令绕过编码器。TMDS 和 TMDS\_REPEAT 命令在传递给输出移位寄存器之前先进行 TMDS 编码。NOP 命令不包含数据，因此其是否绕过编码器属于超出本数据手册范围的哲学性问题。

EXPAND\_SHIFT 寄存器中每个字段均有两个副本。带有 RAW\_ 前缀的字段用于 RAW 和 RAW\_REPEAT 命令。所有其他命令使用带有 ENC\_ 前缀的字段，这些字段通过编码器传递。例如，在 DVI 中，使用 RAW\_REPEAT 命令的 TMDS 控制符号可能不会被移位。使用 TMDS 命令的像素数据可能一次移出一个像素，因此具有分组移位控制非常实用。

EXPAND\_SHIFT.ENC\_N\_SHIFTS 和 EXPAND\_SHIFT.RAW\_N\_SHIFTS 字段分别控制编码命令和原始命令的扩展移位寄存器的重填频率。x\_REPEAT 命令忽略这些字段，因为它们从不从 FIFO 重填，且功能类似于控制输出移位寄存器的 CSR.N\_SHIFTS 字段。

命令扩展器每个周期仅能从数据 FIFO 弹出一次，因此频繁使用命令（特别是 NOP）

命令) 会影响 HSTX 的吞吐量。对于在每个周期均从 HSTX 输出的用例，需将输出移位寄存器配置为 CSR.N\_SHIFTS > 1。原因在于命令扩展器无法在弹出命令的周期输出数据，导致扩展移位寄存器至少空闲一个周期。

### 12.11.6. PIO与HSTX耦合模式

HSTX 最多可连接 8 个 PIO 引脚输出至位交叉开关。仅当 `clk_hstx`直接连接至 `clk_sys`时才使用位交叉开关 (CLK\_HSTX\_CTRL.AUXSRC 必须选择 `clk_sys`)。

#### 注意

仅使两个时钟以相同频率运行是不够的。您必须直接选择 `clk_sys`。

要启用耦合模式，请设置 CSR.COUPLLED\_MODE。同一寄存器中的COUPLED\_SEL字段选择要与 HSTX 耦合的 PIO 实例（编号 0至2）。启用耦合模式时，所选 PIO 实例中编号12至19的 IO 输出将在位交叉开关 PSEL\_N和 PSEL\_P索引31:24处出现，从位交叉开关的视角替代输出移位寄存器的最高有效8位。

该模式允许 PIO 程序利用 HSTX 的 DDR 输出。您可以使用此模式以完整系统时钟速率驱动时钟，或以半系统时钟周期分辨率定位时钟转换相对于数据转换的位置。

用于耦合模式的 PIO 输出始终是对应 GPIO 驱动的引脚输出中的第19位至第12位，且独立于 GPIOBASE。当 `GPIOBASE`为0时，用于耦合模式的 PIO 输出即是通常出现在 `HSTX`引脚上的输出。当 `GPIOBASE`为16时，该设置使用的PIO输出对应GPIO 28至35。

启用耦合模式不会以任何方式影响PIO的操作。

通过HSTX耦合模式接口输出相较于直接由PIO输出到引脚存在一个额外的系统时钟周期延迟。

### 12.11.7. 控制寄存器列表

控制寄存器起始于基地址 `0x400c0000` (在SDK中定义为HSTX\_CTRL\_BASE)。它们通过异步总线交叉访问，因此每次总线访问耗时多个时钟周期，具体时长取决于 `clk_sys`与 `clk_hstx`的比例。

表 1253。HSTX\_CTRL 寄存器列表

偏移量	名称	说明
0x00	CSR	
0x04	BIT0	输出位0的数据控制寄存器
0x08	BIT1	输出位1的数据控制寄存器
0x0c	BIT2	输出位2的数据控制寄存器
0x10	BIT3	输出位3的数据控制寄存器
0x14	BIT4	输出位4的数据控制寄存器
0x18	BIT5	输出位5的数据控制寄存器
0x1c	BIT6	输出位6的数据控制寄存器
0x20	BIT7	输出位7的数据控制寄存器
0x24	EXPAND_SHIFT	配置命令扩展器内部的可选移位器
0x28	EXPAND_TMDS	配置命令扩展器内部的可选TMDS编码器 扩展器

## HSTX\_CTRL: CSR寄存器

偏移: 0x00

表1254. CSR  
寄存器

位	描述	类型	复位
31:28	<p><b>CLKDIV:</b> 生成时钟的周期，单位为HSTX时钟周期。 可以为奇数或偶数。生成的时钟仅在移位寄存器移位的周期内前进。</p> <p>例如，clkdiv为5时，每5个HSTX时钟周期（或每10个半时钟周期）生成一个完整的输出时钟周期。</p> <p>CLKDIV值为0时，映射为16个HSTX时钟周期的周期。</p>	读写	0x1
27:24	<p><b>CLKPHASE:</b> 设置生成时钟的初始相位。</p> <p>CLKPHASE为0表示时钟初始为低电平，第一个上升沿发生在生成时钟半周期之后（即clk_hstx的CLKDIV/2个周期）。CLKPHASE每增加1，将使初始时钟相位提前半个clk_hstx周期。例如，当CLKDIV=2且CLKPHASE=1时：</p> <ul style="list-style-type: none"> <li>* 时钟初始为低电平</li> <li>* 第一个上升沿将在首次数据激活后0.5个clk_hstx周期出现</li> <li>* 第一个下降沿将在首次数据激活后1.5个clk_hstx周期出现</li> </ul> <p>此配置适合以clk_hstx比特率串行化，并采用中心对齐DDR时钟。</p> <p>当通过清除CSR_EN停止HSTX时，时钟发生器将恢复到由CLKPHASE字段配置的初始相位。</p> <p>注意：CLKPHASE必须严格小于CLKDIV的两倍（即一个完整周期），否则其操作行为未定义。</p>	读写	0x0
23:21	保留。	-	-
20:16	<p><b>N_SHIFTS:</b> 在从FIFO重新填充移位寄存器之前移位的次数。（计数的是移位的次数，不是总移位距离。）</p> <p>寄存器值为0表示移位32次。</p>	读写	0x05
15:13	保留。	-	-
12:8	<p><b>SHIFT:</b> 每个周期对移位寄存器右旋转的位数。</p> <p>使用旋转而非移位可通过从32中减去左移位量来模拟左移。当SHIFT与N_SHIFTS的乘积大于32时，该机制还允许数据重复。</p>	读写	0x06
7	保留。	-	-
6:5	<b>COUPLED_SEL:</b> 选择用于耦合模式操作的PIO端口。	读写	0x0

位	描述	类型	复位
4	<p><b>COUPLED_MODE</b>: 启用PIO到HSTX的1:1连接。为使该同步接口正常工作，HSTX必须直接由系统时钟驱动（而非仅由其他相同频率的时钟源驱动）。</p> <p>启用COUPLED_MODE后，BITx_SEL_P和SEL_N中24至31号索引将从8位PIO至HSTX路径中选择位，而非移位寄存器的位。索引0至23仍将按常规方式索引移位寄存器。</p> <p>连接到PIO至HSTX总线的PIO输出，正是当相关引脚的FUNCSEL设置为PIO而非HSTX时，该引脚上所对应的输出。</p> <p>例如，如果HSTX连接至GPIO 12至19，则在耦合模式激活时，PIO输出12至19将连接到HSTX。</p>	读写	0x0
3:2	保留。	-	-
1	<p><b>EXPAND_EN</b>: 启用命令扩展器。当设置为0时，原始FIFO数据直接传递至输出移位寄存器。当设置为1时，命令扩展器可在FIFO与移位寄存器之间执行诸如运行长度解码等简单操作。</p> <p>当EN已设置时，请勿更改CXPD_EN。可以在设置EN的同时安全地设置CXPD_EN。</p>	读写	0x0
0	<p><b>EN</b>: 当EN为1时，HSTX将按照FIFO中的数据顺序移出数据。只要存在数据，HSTX移位寄存器就会每个时钟周期移位一次，且从FIFO弹出的频率由SHIFT与SHIFT_THRESH的比值决定。</p> <p>当EN为0时，FIFO不会被弹出。只要EN为低电平，移位计数器和时钟发生器均被复位至初始状态。请注意，时钟发生器的初始相位可由CLKPHASE字段配置。</p> <p>一旦HSTX重新启用且数据被推入FIFO，生成时钟的第一个上升沿将于首个数据信号启动后半个周期出现。</p>	读写	0x0

## HSTX\_CTRL: BIT0、BIT1、...、BIT6、BIT7寄存器

偏移: 0x04、0x08、...、0x1c、0x20

### 描述

用于输出位的数据显示控制寄存器 n

表1255。BIT0、  
BIT1、...、BIT6、BIT7  
寄存器

位	描述	类型	复位
31:18	保留。	-	-
17	<b>CLK</b> : 请将此输出连接至生成的时钟，而非数据移位寄存器。如果该位被设置，SEL_P 和 SEL_N 将被忽略，但仍可设置 INV 以生成反相时钟。	读写	0x0
16	<b>INV</b> : 反转此数据输出（逻辑非）	读写	0x0
15:13	保留。	-	-
12:8	<b>SEL_N</b> : 选择移位寄存器数据位，用于 HSTX 时钟周期的后半部分	读写	0x00

位	描述	类型	复位
7:5	保留。	-	-
4:0	<b>SEL_P</b> : 选择移位寄存器数据位，用于 HSTX 时钟周期的前半部分	读写	0x00

## HSTX\_CTRL: EXPAND\_SHIFT 寄存器

偏移量: 0x24

### 说明

配置命令扩展器内部的可选移位器

表 1256。  
EXPAND\_SHIFT  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28:24	<b>ENC_N_SHIFTS</b> : 当前命令为编码数据命令（如 TMDS）时，消耗移位寄存器中数据的次数，之后从 FIFO 重新填充寄存器。寄存器值为 0 表示移位 32 次。	读写	0x01
23:21	保留。	-	-
20:16	<b>ENC_SHIFT</b> : 当前命令为编码数据命令（如 TMDS）时，每次向输出移位器传送数据时，移位寄存器向右循环移位的位数。	读写	0x00
15:13	保留。	-	-
12:8	<b>RAW_N_SHIFTS</b> : 当前命令为原始数据命令时，消耗移位寄存器中数据的次数，之后从 FIFO 重新填充寄存器。寄存器值为 0 表示移位 32 次。	读写	0x01
7:5	保留。	-	-
4:0	<b>RAW_SHIFT</b> : 当当前命令为原始数据命令时，每次将数据推送至输出移位器时，移位寄存器右旋转的位数。	读写	0x00

## HSTX\_CTRL: EXPAND\_TMDS 寄存器

偏移: 0x28

### 说明

配置命令扩展器内的可选 TMDS 编码器

表 1257。  
EXPAND\_TMDS  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:21	<b>L2_NBITS</b> : 第 2 通道 TMDS 编码器的有效数据位数，从旋转后数据的第 7 位起计数。字段值 0 → 7 对应编码长度为 1 → 8 位。	读写	0x00
20:16	<b>L2_ROT</b> : 应用于第 2 通道 TMDS 编码器前的当前移位器数据的右旋转位数。	读写	0x00
15:13	<b>L1_NBITS</b> : 第 1 通道 TMDS 编码器的有效数据位数，从旋转后数据的第 7 位起计数。字段值 0 → 7 对应编码长度为 1 → 8 位。	读写	0x00
12:8	<b>L1_ROT</b> : 应用于第 1 通道 TMDS 编码器前的当前移位器数据的右旋转位数。	读写	0x00

位	描述	类型	复位
7:5	<b>L0_NBITS</b> : 通道0 TMDS编码器的有效数据位数，从旋转数据的第7位起算。字段值 0 → 7 对应编码长度为 1 → 8 位。	读写	0x0
4:0	<b>L0_ROT</b> : 施加于当前移位器数据的右旋转量，作用于通道0 TMDS编码器之前。	读写	0x00

### 12.11.8. FIFO寄存器列表

FIFO寄存器的起始基址为 **0x50600000** (在SDK中定义为HSTX\_FIFO\_BASE)。

表1258。HSTX\_FIFO 寄存器列表

偏移量	名称	说明
0x0	<b>STAT</b>	FIFO状态
0x4	<b>FIFO</b>	FIFO写访问

### HSTX\_FIFO：STAT寄存器

偏移量: 0x0

#### 说明

FIFO状态

表1259。STAT 寄存器

位	描述	类型	复位
31:11	保留。	-	-
10	<b>WOF</b> : FIFO已满时仍有写入。写入1以清除。	WC	0x0
9	<b>空</b>	只读	-
8	<b>满</b>	只读	-
7:0	<b>LEVEL</b>	只读	0x00

### HSTX\_FIFO：FIFO寄存器

偏移量: 0x4

表1260。FIFO 寄存器

位	描述	类型	复位
31:0	FIFO写访问	WF	0x00000000

## 12.12. TRNG

### 12.12.1. 概述

RP2350包含基于Arm IP的真正随机数生成模块。支持以下功能：

- 符合FIPS Publication 140-2、BSI AIS-31及NIST SP 800-90B标准
- 当核心运行于150 MHz时，产生约7.5 kb/s的熵

应请求，TRNG模块生成一块192位的熵数据，通过自动处理来自TRNG模块内部环形振荡器（ROSC）的一系列周期性采样产生。

TRNG模块内的ROSC为自由振荡器，且与RP2350系统时钟无直接连接，因此ROSC通常异步于系统时钟运行。

采集足够数量的样本后，TRNG模块完成生成过程，并将随机数输出至EHR\_DATA[x]结果寄存器。

更多详情，请参见ARM IP - 真随机数生成器

### 12.12.2. 配置

TRNG模块包含三种内置熵检测机制：复位时默认全部启用，无需显式激活。

您可以通过以下方式配置TRNG模块：

- 通过选择四种ROSC链长之一，配置ROSC的频率，详见TRNG\_CONFIG。
- 根据系统时钟周期配置ROSC采样周期，详见SAMPLE\_CNT1。

由于系统时钟通常远快于ROSC，采样周期预计至少为数十个系统时钟周期。

鉴于TRNG ROSC特性及系统时钟频率因每个TRNG IP模块的实现而异，Arm制定了TRNG特征化程序，以确定每个SoC设计中最合适的ROSC链长度与采样频率设置。有关该特征化程序的详细信息，请参见ARM TrustZone真随机数生成器。

RP2350 TRNG模块的软件驱动不采用标准方案（参见第12.12.4节），因此软件不会配置Arm特征化程序所提供的ROSC长度及采样计数设置。

配置TRNG模块时，请遵循以下原则：

- 随着平均生成时间的增加，结果质量提升，熵校验失败次数减少。
- 较低的采样计数虽能降低平均生成时间，但会增加NIST测试失败和熵校验失败的概率。

为了获得约2毫秒平均生成时间的可接受结果，请将ROSC链长度设置为0或1，样本计数设置为20至25。

较大的样本计数设置（例如100）将导致平均生成时间相应增加。这些设置虽能显著减少NIST测试失败及熵检测失败的发生，但无法完全消除。生成结果有时可能耗费异常长的时间。

### 12.12.3. 操作

要启动TRNG生成，请在RND\_SOURCE\_ENABLE寄存器中设置 RND\_SRC\_EN位。TRNG将持续运行，直到：

- 成功完成一次随机数生成。
- 一个或多个内部熵检测机制指示运行失败。

无论哪种情况，您均可从RNG\_ISR寄存器读取结果状态。

要生成TRNG块中断，请设置RNG\_IMR寄存器中的相关位。使用RNG\_ICR清除活动中断状态位。

在成功生成之前，EHR\_DATA[x]寄存器读取值为0，因此CPU在生成期间无法读取随机数结果，

生成成功后，读取最后的结果寄存器EHR\_DATA[5]以清除所有结果寄存器。如果结果未通过熵检查，则不会提供任何结果，且EHR\_DATA[x]寄存器均读取为0。

TRNG生成后且不使用时，应清除 RND\_SRC\_EN位。

## 12.12.4. 注意事项

TRNG模块生成随机数的过程非确定性。

尽管生成随机数所需的众数和平均时间非常接近，生成过程有时却会大幅延长：超过平均时间100倍以上。任何未通过熵检查的运行结果均被丢弃，需重新进行生成。

您应在系统设计中考虑这些不可预测的生成时间。例如，您可以预生成一小批随机数池，在池中有空间时启动后续生成。

为简化设计并提升时序可预测性，RP2350 bootrom及SDK TRNG模块驱动采用了替代方案。所采用的方法论可通过以下链接获取。然而，RP2350中的TRNG模块并不排除按照Arm文档规范使用该模块。

### 12.12.4.1. Bootrom

bootrom将原始的TRNG ROSC样本（即TRNG随机源）直接输入到硬件SHA-256加速器。它绕过了TRNG中的所有内部检测和调节。SHA-256是一种强健的哈希函数，能够避免TRNG部分调节逻辑的缺陷，尤其是冯·诺依曼去相关器。

bootrom具有若干严格约束以指导其实现选择，最显著的是：bootrom必须能够成功启动。它无法承受轮询TRNG以等待随机数出现的不确定时长。复杂的错误处理同样不可取。

第5章中提供了bootrom源码链接。请参考源代码以了解每次启动随机数生成的具体实现，位于[varm\\_boot\\_path.c](#)。

由于多种实现限制，A2 bootrom 的 TRNG 代码采用汇编语言编写，可能不够直观。以下摘录自 A1 bootrom 源代码，经过轻微编辑以提升可读性：

```
// 引导随机数生成器 (RNG) 通过对大量 TRNG ROSC 样本进行流式处理而得到
// 并送入 SHA-256。BOOT_TRNG_SAMPLE_BLOCKS 表示要哈希的 SHA-256 块数，

const int BOOT_TRNG_SAMPLE_BLOCKS = 25;

// TRNG 软复位后需固定延迟 trng_hw->trng_sw_reset = -
1u;
(void)trng_hw->trng_sw_reset;
(void)trng_hw->trng_sw_reset;
// 通过写入 START 位初始化 SHA 内部状态
sha256_hw->csr = SHA256_CSR_RESET | SHA256_CSR_START_BITS;

// 每周期将一个 ROSC 位采样到 EHR，前提是 CPU 能跟上。（还有更多的//亚稳态事件，它们是次要的噪声源）

trng_hw->sample_cnt1 = 0;
// 禁用检查并绕过去相关器，以流式传输原始TRNG ROSC样本：
trng_hw->trng_debug_control = -1u;
// 如果ROSC尚未启动，则启动ROSC
trng_hw->rnd_source_enable = -1u;
// 清除所有中断（包括EHR_VLD）——稍后在为RCP加种子时会检查此项。

trng_hw->rng_icr = -1u;

// 每个半块 (192个样本) 大约需要235个周期，因此每块约470个周期：
for (int half_blocks = 0; half_blocks < 2 * BOOT_TRNG_SAMPLE_BLOCKS; ++half_blocks) {

 // 等待192个ROSC样本填满EHR，该过程应持续固定时间：
 while (trng_hw->trng_busy)
```

```

;

// 将6个EHR字复制至SHA-256，外加垃圾数据（RND_SOURCE_ENABLE和// SAMPLE_C
NT1），用于填充至SHA-256块的一半。这意味着//我们可以在读取EHR时避免检查SHA-256是
否就绪，从而更早重新开始采样。（每写入16个字后，SHA-256会非就绪状态持续57个时钟周期
。）

io_ro_32 *src = &trng_hw->ehr_data[0];
io_wo_32 *dst = &sha256_hw->wdata;
for (int i = 0; i < 8; ++i) {
 *dst = src[i];
}

// TRNG已重新启动采样，因我们刚读取完最后一个EHR字。提取部分SHA正在进行的位数，用
以调节链长，以降低注入锁定的可能性：

trng_hw->trng_config = sha256_hw->sum[0];
}

// 等待SHA结果——若跳过此步，将获上一区块的摘要。注意//若写入字数%m 16 ≠ 0，此条件将永远
不成立。
while (!(sha256_hw->csr & SHA256_CSR_SUM_VLD_BITS))
;

// 每次核心0重置时，每次启动的随机数都会发生变化（调试器例外，跳过只读存储器）。若此
为问题，用户可将该启动随机数采样保存于主SRAM中的持久变量。

for (int i = 0; i < 4; ++i) {
 bootrom->always.boot_random.e[i] = sha256_hw->sum[4 + i];
}

trng_hw->trng_config = 0;
// 停止ROSC以节省功耗
trng_hw->rnd_source_enable = 0;

```

bootrom 在上述代码执行前，通过复位（RESETS）立即重置了 SHA-256 和真随机数生成器（TRNG）。该代码通常在系统ROSC以约12 MHz初始启动频率运行时执行。代码完成后，256位结果可在 SUM0 至 SUM7 寄存器中读取。

该代码并非最佳编程实践：例如，它向 TRNG\_DEBUG\_CONTROL 寄存器的保留位写入了全1。其编写过程紧密参照硬件实现。上述代码仅用于记录bootrom在启动时生成随机数的方法，此随机数通过[get\\_sys\\_info\(\)](#)只读存储器 API 获取，每次启动可用一次，同时用于初始化 RCP 盐寄存器（第 3.6.3.1 节）。

#### 12.12.4.2. SDK

该 [pico\\_rand](#) 库以 TRNG 作为其熵源之一。其以类似 bootrom 的方式，从 TRNG ROSC 流式传输原始 ROSC 采样。它应用 [xoroshiro128\\*\\*](#) 和 [splitmix64\(\)](#) 伪随机数生成函数对输出进行调节。

#### 12.12.5. 寄存器列表

TRNG 控制寄存器地址为 [0x400f0000](#)（SDK 中定义为 TRNG\_BASE）。

表1261：TRNG寄  
存器列表

偏移量	名称	说明
0x100	RNG_IMR	中断屏蔽。

偏移量	名称	说明
0x104	RNG_ISR	随机数发生器状态寄存器。若对应 RNG_IMR 位未屏蔽，则会触发中断。
0x108	RNG_ICR	中断/状态位清除寄存器。
0x10c	TRNG_CONFIG	选择反相器链长度。
0x110	TRNG_VALID	192位集合指示。
0x114	EHR_DATA0	随机数生成器所收集的位。
0x118	EHR_DATA1	随机数生成器所收集的位。
0x11c	EHR_DATA2	随机数生成器所收集的位。
0x120	EHR_DATA3	随机数生成器所收集的位。
0x124	EHR_DATA4	随机数生成器所收集的位。
0x128	EHR_DATA5	随机数生成器所收集的位。
0x12c	RND_SOURCE_ENABLE	随机源的使能信号。
0x130	SAMPLE_CNT1	采样随机位之间的时钟计数。
0x134	AUTOCORR_STATISTIC	有关自相关测试激活的统计数据。
0x138	TRNG_DEBUG_CONTROL	调试寄存器。
0x140	TRNG_SW_RESET	在随机数生成器模块内生成内部软件复位。
0x1b4	RNG_DEBUG_EN_INPUT	使能随机数生成器调试模式。
0x1b8	TRNG_BUSY	随机数生成器忙碌指示。
0x1bc	RST_BITS_COUNTER	复位随机数生成器中收集位的计数器。
0x1c0	RNG_VERSION	显示真随机数生成器的版本设置。
0x1e0	RNG_BIST_CNTR_0	收集的内建自检（BIST）结果。
0x1e4	RNG_BIST_CNTR_1	收集的内建自检（BIST）结果。
0x1e8	RNG_BIST_CNTR_2	收集的内建自检（BIST）结果。

## 真随机数生成器：RNG\_IMR 寄存器

偏移：0x100

### 描述

中断屏蔽。

表1262: RNG\_IMR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>VN_ERR_INT_MASK:</b> 置1以屏蔽（禁用）该中断：不会产生中断。请参阅 RNG_ISR 以了解该中断的详细说明。	读写	0x1
2	<b>CRNGT_ERR_INT_MASK:</b> 置为 1 以屏蔽（禁用）该中断：中断将不会产生。请参阅 RNG_ISR 以了解该中断的详细说明。	读写	0x1
1	<b>AUTOCORR_ERR_INT_MASK:</b> 置为 1 以屏蔽（禁用）该中断：中断将不会产生。请参阅 RNG_ISR 以了解该中断的详细说明。	读写	0x1
0	<b>EHR_VALID_INT_MASK:</b> 置为 1 以屏蔽（禁用）该中断：中断将不会产生。请参阅 RNG_ISR 以了解该中断的详细说明。	读写	0x1

## TRNG: RNG\_ISR 寄存器

偏移量: 0x104

### 描述

随机数发生器状态寄存器。若对应 RNG\_IMR 位未屏蔽，则会触发中断。

表1263: RNG\_ISR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>VN_ERR:</b> 1 表示发生 von Neumann 错误。当连续采集的 32 位比特全部相同（全为零或全为一）时，即发生 von Neumann 错误。	只读	0x0
2	<b>CRNGT_ERR:</b> 1 表示 RNG 测试中的 CRNGT 失败。当连续两个采集的 16 位块相同时，即判定为失败。	只读	0x0
1	<b>AUTOCORR_ERR:</b> 1 表示自相关测试连续失败四次。 设置后，RNG 将停止工作，直至下一次复位。	只读	0x0
0	<b>EHR_VALID:</b> 1 表示已在 RNG 中收集 192 位，且可读取。	只读	0x0

## TRNG: RNG\_ICR 寄存器

偏移: 0x108

### 说明

中断/状态位清除寄存器。

表1264: RNG\_ICR  
寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>VN_ERR:</b> 写入 1 以清除 RNG_ISR 中相应位。	读写	0x0
2	<b>CRNGT_ERR:</b> 写入 1 以清除 RNG_ISR 中相应位。	读写	0x0
1	<b>AUTOCORR_ERR:</b> 无法通过软件清除！仅 RNG 复位可清除此位。	读写	0x0
0	<b>EHR_VALID:</b> 写入 1 以清除 RNG_ISR 中相应位。	读写	0x0

## TRNG: TRNG\_CONFIG 寄存器

偏移: 0x10c

### 描述

选择反相器链长度。

表1265:  
TRNG\_CONFIG  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1:0	<b>RND_SRC_SEL:</b> 选择环形振荡器（熵源）中反相器数量（四种可能选择之一）。数值越大，反相器链长度越长。	读写	0x0

## TRNG: TRNG\_VALID 寄存器

偏移量: 0x110

### 描述

192 位集合指示。

表1266。  
TRNG\_VALID寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>EHR_VALID:</b> 1表示随机数生成器（RNG）中的比特收集已完成，可从EHR_DATA寄存器读取数据。	只读	0x0

## TRNG: EHR\_DATA0、EHR\_DATA1、...、EHR\_DATA4、EHR\_DATA5寄存器

偏移量: 0x114, 0x118, ..., 0x124, 0x128

### 描述

随机数生成器所收集的位。

表1267。  
EHR\_DATA0、EHR\_DATA1、...、EHR\_DATA4、EHR\_DATA5寄存器

位	描述	类型	复位
31:0	熵保持寄存器中位于[(32*(i+1))-1:(32*i)]范围的位。	只读	0x00000000

## TRNG: RND\_SOURCE\_ENABLE寄存器

偏移量: 0x12c

### 描述

随机源的使能信号。

表1268。  
RND\_SOURCE\_ENABLE寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>RND_SRC_EN:</b> * 1 - 熵源已启用。 * 0 - 熵源已禁用。	读写	0x0

## TRNG: SAMPLE\_CNT1寄存器

偏移量: 0x130

### 描述

采样随机位之间的时钟计数。

表1269。  
SAMPLE\_CNT1寄存器

位	描述	类型	复位
31:0	<b>SAMPLE_CNTR1:</b> 设置两个连续环形振荡器采样之间的rng_clk周期数。  注意：如果绕过冯·诺依曼去相关器，采样计数器的最小值不得小于十七。 。	读写	0x0000ffff

## TRNG: AUTOCORR\_STATISTIC寄存器

偏移量: 0x134

### 描述

有关自相关测试激活的统计数据。

表1270。  
AUTOCORR\_STATIST  
IC寄存器

位	描述	类型	复位
31:22	保留。	-	-
21:14	<b>AUTOCORR_FAILS:</b> 统计每次自相关测试失败的次数。对该寄存器的任何写操作均会重置计数器。当任一计数器达到限制时，停止收集统计数据。	读写	0x00

位	描述	类型	复位
13:0	<b>AUTOCORR_TRYS</b> : 统计每次启动自相关测试的次数。对该寄存器的任何写操作均会重置计数器。当任一计数器达到限制时，停止收集统计数据。	读写	0x0000

## TRNG: TRNG\_DEBUG\_CONTROL 寄存器

偏移量: 0x138

### 描述

调试寄存器。

表1271。  
TRNG\_DEBUG\_CONTROL  
ROL 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>AUTO_CORRELATE_BYPASS</b> : 设置后，TRNG模块中的自相关测试将被绕过。	读写	0x0
2	<b>TRNG_CRNGT_BYPASS</b> : 设置后，绕过随机数生成器（RNG）中的CRNGT测试。	读写	0x0
1	<b>VNC_BYPASS</b> : 设置后，绕过冯·诺依曼均衡器（包括32位连续位测试）。  不适用	读写	0x0
0	保留。	-	-

## TRNG: TRNG\_SW\_RESET 寄存器

偏移: 0x140

### 描述

在随机数生成器模块内生成内部软件复位。

表1272  
TRNG\_SW\_RESET  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>TRNG_SW_RESET</b> : 写入1至该寄存器将导致内部RNG复位。	读写	0x0

## TRNG: RNG\_DEBUG\_EN\_INPUT 寄存器

偏移: 0x1b4

### 描述

使能随机数生成器调试模式。

表1273  
RNG\_DEBUG\_EN\_INPUT  
UT 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>RNG_DEBUG_EN</b> : * 1 - 启用调试模式。  * 0 - 禁用调试模式。	读写	0x0

## TRNG: TRNG\_BUSY 寄存器

偏移: 0x1b8

### 描述

RNG忙碌状态指示。

表1274  
TRNG\_BUSY寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>TRNG_BUSY:</b> 反映rng_busy状态。	只读	0x0

## TRNG: RST\_BITS\_COUNTER 寄存器

偏移: 0x1bc

### 描述

复位随机数生成器中收集位的计数器。

表1275。  
RST\_BITS\_COUNTER  
寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	<b>RST_BITS_COUNTER:</b> 向该地址写入任意值将重置位计数器及RNG有效寄存器。RND_SOURCE_ENABLE寄存器必须未设置，重置操作方能生效。	读写	0x0

## TRNG: RNG\_VERSION 寄存器

偏移: 0x1c0

### 描述

显示真随机数生成器的版本设置。

表1276。  
RNG\_VERSION  
寄存器

位	描述	类型	复位
31:8	保留。	-	-
7	<b>RNG_USE_5_SBOXES:</b> * 1 - 使用5个SBOX的AES。 * 0 - 使用20个SBOX的AES	只读	0x0
6	<b>RESEEDING_EXISTS:</b> * 1 - 存在。 * 0 - 不存在	只读	0x0
5	<b>KAT_EXISTS:</b> * 1 - 存在。 * 0 - 不存在	只读	0x0
4	<b>PRNG_EXISTS:</b> * 1 - 存在。 * 0 - 不存在	只读	0x0
3	<b>TRNG_TESTS_BYPASS_EN:</b> * 1 - 存在。 * 0 - 不存在	只读	0x0
2	<b>AUTOCORR_EXISTS:</b> * 1 - 存在。 * 0 - 不存在	只读	0x0
1	<b>CRNGT_EXISTS:</b> * 1 - 存在。 * 0 - 不存在	只读	0x0
0	<b>EHR_WIDTH_192:</b> * 1 - 192位EHR。 * 0 - 128位EHR	只读	0x0

## TRNG: RNG\_BIST\_CNTR\_0 寄存器

偏移：0x1e0

### 描述

收集的内建自检（BIST）结果。

表1277  
RNG\_BIST\_CNTR\_0  
寄存器

位	描述	类型	复位
31:22	保留。	-	-
21:0	<b>ROSC_CNTR_VAL</b> : 反映 RNG BIST 计数器的结果。	只读	0x000000

## TRNG: RNG\_BIST\_CNTR\_1 寄存器

偏移量：0x1e4

### 描述

收集的 BIST 结果。

表1278  
RNG\_BIST\_CNTR\_1  
寄存器

位	描述	类型	复位
31:22	保留。	-	-
21:0	<b>ROSC_CNTR_VAL</b> : 反映 RNG BIST 计数器的结果。	只读	0x000000

## TRNG: RNG\_BIST\_CNTR\_2 寄存器

偏移量：0x1e8

### 描述

收集的 BIST 结果。

表1279  
RNG\_BIST\_CNTR\_2  
寄存器

位	描述	类型	复位
31:22	保留。	-	-
21:0	<b>ROSC_CNTR_VAL</b> : 反映 RNG BIST 计数器的结果。	只读	0x000000

## 12.13. SHA-256 加速器

RP2350 采用了 SHA-256 哈希算法的实现，该算法依据 NIST 发布的 FIPS 180-4 标准定义。哈希算法对任意长度的数据流，即“消息”，进行摘要计算，生成固定长度的结果，即“哈希值”。在SHA-256中，结果始终为256位。哈希算法设计具有以下特性：

- 已知哈希值时，不可能（或计算上极为困难）恢复原始消息。
- 对原始消息的细微更改通常会导致哈希值发生显著变化。
- 已知某消息及其哈希值时，不可能（或计算上极为困难）生成具有相同哈希值的不同消息。

这些特性使哈希算法在防止意外位翻转和恶意篡改时，用于验证数据完整性。

使用RP2350 SHA-256加速器计算SHA-256时，应执行以下步骤：

1. 向CSR.START寄存器写入1以初始化算法状态。
2. 将消息写入WDATA寄存器，写入期间轮询CSR.WDATA\_RDY状态。

3. 根据第12.13.1节的描述，向WDATA寄存器写入额外的尾部和填充数据。
4. 轮询CSR.SUM\_VLD寄存器，等待最后一个数据块被处理完成。
5. 从起始于 SUM0 的 8 个只读结果寄存器中读取 256 位结果。

### 12.13.1. 消息填充

根据 FIPS 180-4 《安全哈希标准》中描述的标准 SHA-256 方法对消息内容进行填充：在消息后附加单个比特 **1**，随后是若干个 **0** 比特，然后附加表示消息比特数的 64 位计数。因此，对于长度为 L 比特的消息 M，填充后的消息应为：

1. 消息 **M**
2. **1**
3. **k** 个零比特，其中 **k** 为满足以下方程的最小非负整数解： $L + 1 + k = 448 \bmod 512$
4. 一个 64 位区块，表示消息长度 **L**（二进制格式）

例如，8 位 ASCII 消息 **abc** 的长度为 24 比特。其填充方式为：先附加 **1** 比特，再附加  $448 - (24 + 1) = 423$  个 **0** 比特，最后附加表示消息长度的 64 位值，如下所示：

```
01100001 01100010 01100011 1 00000000 000...0 00000000 000...0 00011000
|-----消息-----| 1 |--423 0 bits--| |----64 bit len----|
```

### 12.13.2. 吞吐量

SHA-256 每次处理一个 512 位的数据块。这需要向 WDATA 寄存器写入 16 次 32 位数据、32 次 16 位数据或 64 次 8 位数据。一次 APB 寄存器写入耗费 4 个周期，因此写入一个数据块至少需要 64 个系统时钟周期。

数据块传输完成后，SHA 核心还需 57 个周期以完成该数据块的摘要计算。当 CSR.WDATA\_RDY 变为低电平时，期间不得向 WDATA 寄存器写入数据。

因此最大吞吐量为每 121 个系统时钟周期处理一个数据块，或每个周期 0.53 字节。在 150 MHz 系统时钟下，该吞吐量为 79.3 MB/s。此吞吐量通过使用 DMA 进行 32 位传输实现。使用较窄的数据传输宽度或在处理器传输数据时轮询 CSR.WDATA\_RDY 标志，均会降低吞吐量。

### 12.13.3. 数据大小与字节序

数据以 512 位的消息块形式发送，填充方式详见第 12.13.1 节。SHA-256 加速器针对每个输入块更新其 256 位输出状态。SHA-256 算法以大端消息字定义，但该加速器通过 CSR.BSWAP 提供字节交换功能，以支持小端数据。**BSWAP** 默认开启。

欲了解更多信息，请参见寄存器描述。

**WDATA** 支持 8 位、16 位及 32 位写入。总线接口在将数据传输至 SHA-256 算法核心之前，会将 8 位和 16 位写入累积于 32 位移位寄存器中。因此，在混合不同大小的写入时必须谨慎，单次写入若导致移位寄存器位宽从低于 32 位提升至高于 32 位，将无声丢弃数据。为避免此问题，建议在单个 SHA-256 消息块（64 字节）中不要混用 **WDATA** 写入大小。

### 12.13.4. DMA DREQ 接口

该块可请求 DMA 控制器一次传送完整数据块。通过以下方式配置传输大小

**CSR.DMA\_SIZE**, 以确保DMA控制器请求正确数量的传输。

DREQ始终一次请求完整的一个SHA数据块。请勿在非块边界启动DMA传输。

### 12.13.5. 寄存器列表

SHA-256寄存器的基地址为 **0x400f8000** (在SDK中定义为SHA256\_BASE)。

表1280。  
SHA256寄存器列表

偏移量	名称	说明
0x00	<b>CSR</b>	控制与状态寄存器
0x04	<b>WDATA</b>	写数据寄存器
0x08	<b>SUM0</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x0c	<b>SUM1</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x10	<b>SUM2</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x14	<b>SUM3</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x18	<b>SUM4</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x1c	<b>SUM5</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x20	<b>SUM6</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。
0x24	<b>SUM7</b>	256位校验和结果。当CSR_SUM_VLD为0时, 内容未定义。

### SHA256: CSR 寄存器

偏移: 0x00

#### 说明

控制与状态寄存器

表1281。CSR  
寄存器

位	描述	类型	复位
31:13	保留。	-	-

位	描述	类型	复位
12	<p><b>BSWAP:</b> 启用在将32位值提交至SHA消息调度器时的字节交换功能</p> <ul style="list-style-type: none"> <li>◦ 该模块的总线接口以小端序将字节或半字数据组装成消息字，使得在RP2350等小端系统上以不同传输大小对同一缓冲区进行DMA操作时，结果始终一致</li> <li>◦ 然而，在将字节编组到区块时，SHA期望每个消息字中的第一个字节为最高有效字节。为解决此问题，当总线接口累积32位数据（包括一次字写入、两次小端序半字写入或四次小端序字节写入）后，最终值可在传递至实际SHA核心之前进行字节交换。</li> <li>◦ 该功能默认为启用状态，因为预期使用SHA核心对字节缓冲区进行校验和的情况比直接使用预格式化的SHA消息字更为常见。</li> </ul>	读写	0x1
11:10	保留。	-	-
9:8	<p><b>DMA_SIZE:</b> 配置DREQ逻辑以实现正确的DMA数据大小。必须在触发DMA通道之前完成配置。</p> <p>SHA-256核心的DREQ逻辑由于无FIFO，实现一次请求整块数据，数据直接进入核心的消息调度和摘要硬件。因此，使用DMA传输数据时，必须预先配置CSR_DMA_SIZE，以确保每块请求正确数量的传输。</p>	读写	0x2
	枚举值：		
	0x0 → 8BIT		
	0x1 → 16BIT		
	0x2 → 32BIT		
7:5	保留。	-	-
4	<b>ERR_WDATA_NOT_RDY:</b> 当SHA-256核心尚未准备好接收数据（WDATA_READY为低）时发生写操作，此标志被置位。写入1以清除。	WC	0x0
3	保留。	-	-
2	<p><b>SUM_VLD:</b> 若为1，表示寄存器SUM0至SUM7中显示的SHA-256校验和当前有效。</p> <p>首次写入WDATA时信号变低，写入16字后恢复高电平，且当前512位数据块的摘要计算完成。</p>	只读	0x1
1	<p><b>WDATA_RDY:</b> 若为1，表示SHA-256核已准备好通过WDATA寄存器接收更多数据。</p> <p>写入16个字后，该标志将保持低电平57个周期，期间核将完成摘要计算</p> <ul style="list-style-type: none"> <li>◦</li> </ul>	只读	0x1

位	描述	类型	复位
0	<p><b>START:</b> 写1以准备SHA-256核进行新的校验和计算。</p> <p>SUMx寄存器初始化为正确值（前8个素数平方根的小数部分），内部计数器被清零。这会立即使WDATA_RDY和SUM_VLD置为高电平。</p> <p>启动DMA传输到SHA-256核前，必须先写入START，因为该核始终一次请求16次传输（即1个512位块）。</p> <p>此外，DMA通道应配置为16的32位传输的整数倍。</p>	SC	0x0

## SHA256： WDATA寄存器

偏移量: 0x04

### 描述

写数据寄存器

表1282. WDATA  
寄存器

位	描述	类型	复位
31:0	<p>在发出START脉冲并向该寄存器写入16个字数据后，WDATA_RDY将变为低电平，SHA-256核将完成当前512位块的摘要计算。</p> <p>软件负责确保数据被正确填充并以512位块的整数倍进行终止。</p> <p>此后，WDATA_RDY信号将保持高电平，并且可以继续写入更多数据（如有）。</p> <p>该寄存器支持字（word）、半字（halfword）及字节（byte）写入，因而能够支持来自非字对齐缓冲区的DMA传输。每个数据块的总数据量保持不变（16个字，32个半字或64个字节），且块内不得混合使用字节或半字传输。</p>	WF	0x00000000

## SHA256： SUM0寄存器

偏移: 0x08

表1283. SUM0  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256： SUM1寄存器

偏移: 0x0c

表1284. SUM1  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256： SUM2寄存器

偏移: 0x10

表1285. SUM2  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256: SUM3寄存器

偏移: 0x14

表1286. SUM3  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256: SUM4寄存器

偏移: 0x18

表1287. SUM4  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256: SUM5寄存器

偏移: 0x1c

表1288. SUM5  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256: SUM6 寄存器

偏移: 0x20

表 1289. SUM6  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## SHA256: SUM7 寄存器

偏移: 0x24

表 1290. SUM7  
寄存器

位	描述	类型	复位
31:0	256位校验和结果。当CSR_SUM_VLD为0时，内容未定义。	只读	0x00000000

## 12.14. QSPI 存储接口（QMI）

### 12.14.1. 概述

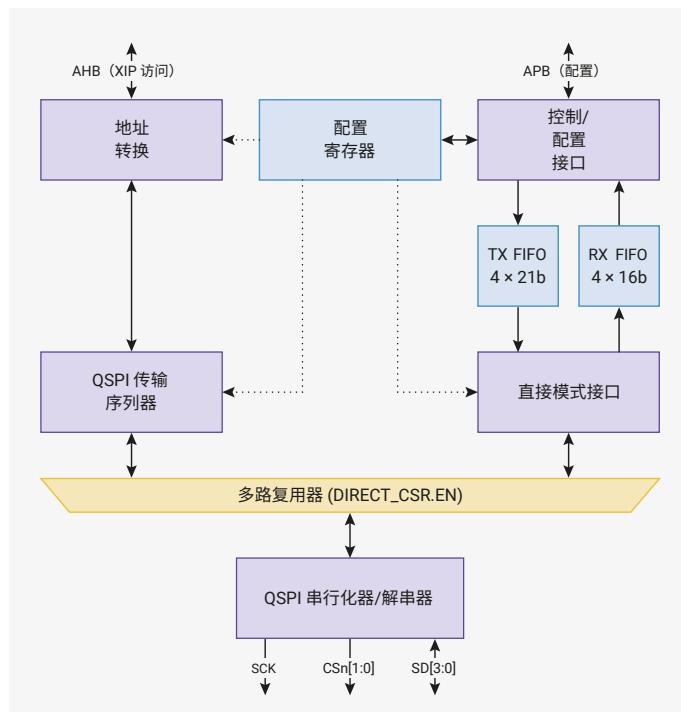
QSPI 存储器接口（QMI）提供对两个外部 QSPI 存储设备的读/写内存映射访问。

RP2350 具有单一的 QMI 实例，嵌入于 XIP 子系统（第 4.4 节），替代了 RP2040 上的 SSI 接口。QMI 支持串行 SPI、双 SPI 和四 SPI 传输，配备两个片选信号及共享的时钟/数据信号。

图129。QMI框图：

AHB访问经过地址转换，拆分为必要的QSPI传输阶段，如命令、地址和数据，通过串行/反串行器与外部QSPI信号接口。

每个设备均设有芯片选择信号，时钟和数据信号为共享信号。此外，直接模式接口可通过一对FIFO发送原始SPI命令，用于编程和配置外部QSPI设备。



每个芯片选择对应16 MB的AHB地址窗口，因此最多支持32 MB外部存储器。芯片选择0拥有专用的外部引脚，其地址映射从 **0x1** 开始**0000000**，芯片选择1作为备用GPIO功能提供，其地址映射从 **0x1100000** 开始。该映射反映于第4.4节所述的无缓存及无缓存加未转换的XIP地址窗口中。

所有时序和SPI命令格式参数均针对每个芯片选择信号配置，且根据地址译码自动采用正确的配置。例如，M0\_TIMING配置芯片选择0的时序参数，M1\_TIMING为芯片选择1对应的同类寄存器。

串行时钟（**SCK**）是系统时钟的任意整数分频，分频范围为1至256。分频系数可随时调整。输入采样时序可按半个系统时钟周期增量调整，以补偿高 **SCK**频率下的时钟至数据延迟。双倍传输速率模式（DTR）通过将 **SCK**频率减半实现，同时保持数据传输速率，且速率最高为每个系统时钟周期4位。

每次访问发出的 **SCK**周期数取决于访问大小，范围从一个字节至一个缓存行。例如，处理器对未缓存的单字节读取操作将在QSPI总线上精确获取一个字节的数据，以避免浪费时间读取不需要的数据。Cache misses 始终以64位QSPI传输方式发出。

QMI可选择性地自动将连续地址的AHB访问链式合并为单一的长QSPI传输。这避免在QSPI总线上发出冗余命令及地址，特别有利于冷代码路径和通过XIP流式硬件（第4.4.3节）流式传输闪存数据。为兼容PSRAM，当链路超过最大片选时间（M0\_TIMING.MAX\_SELECT）或跨越特定的2的幂次地址边界（M0\_TIMING.PAGEBREAK）时，链路可被中断。[第12.14.2.1节](#)对此功能进行了详细说明。

QMI可通过其内置地址转换硬件映射地址：每个片选划分为4个4 MB窗口，其物理基址和孔径大小均以4 KB（一个闪存扇区）为单位配置。这使得闪存程序的运行时地址独立于其存储位置：例如，驻留于闪存存储地址0处的启动加载程序可选择多个闪存驻留程序映像中的一个，这些映像均被链接为在地址 **0x1**上运行。**0000000**这些指令可以直接在原地执行，无需使用位置无关代码。地址转换详见第12.14.4节。

最后，直接模式接口包含于内，适用于软件需要与外部QSPI设备直接通信的场景，例如访问状态寄存器。该接口同样支持串行、双线和四线接口宽度，具体详见第12.14.5节。

## 12.14.2. QSPI 传输

QSPI总线将一个主机（如QMI）连接至多个设备，如串行NOR闪存，包含以下部分：

- 每个设备各一条芯片选择线（**CSn**）
- 一条共享时钟线（**SCK**）
- 最多四条共享数据线（**SD0**至**SD3**）

目前暂无单一规范定义QSPI命令格式。然而，大多数QSPI闪存/SRAM/PSRAM设备采用某些事实标准的命令集。QMI支持这些命令的多数常见变体。

QMI主要作为内存接口，而非通用QSPI外设。尽管直接模式接口（第12.14.5节）允许通过FIFO传递原始数据实现任意QSPI访问，QMI更为优化用于响应AHB读写总线访问的预格式化读写传输。

所有由QMI执行的QSPI读写访问均包含以下五个阶段：

1. 前缀：一个可选的、固定的8位值，用于指示正在执行的SPI命令（在SPI设备数据手册中称为命令前缀或指令前缀）
2. 地址：一个24位字节地址，指定被读写的SPI存储位置，且对应AHB地址的低24位
3. 后缀：一个可选的、固定的8位值，在某些访问模式中紧随地址之后
4. 虚拟：取值为0（SPI）或高阻抗状态（双线/四线SPI）的时钟周期，位于数据信号之前，为SPI设备提供访问起始地址的充分时间
5. 数据：从地址阶段指示的起始地址开始，按顺序字节地址传输内存内容至/自SPI设备

在前缀阶段之前，所选设备的片选信号被置位；在数据阶段结束时，片选信号被复位。

每个阶段包含以位为单位的长度和接口宽度（单/双/四线），通过M0\_RFMT/M1\_RFMT（用于读取）及M0\_WFMT/M1\_WFMT（用于写入）进行配置。每个寄存器的 M0/M1 版本分别用于配置对存储窗口0和1（即两个片选）的访问。该设计允许您透明地使用不同命令格式寻址两个不同的QSPI设备。

图130。一个示例串行读取流程。在8位前缀后，主机发送24位地址，设备自下一个时钟周期开始返回数据。

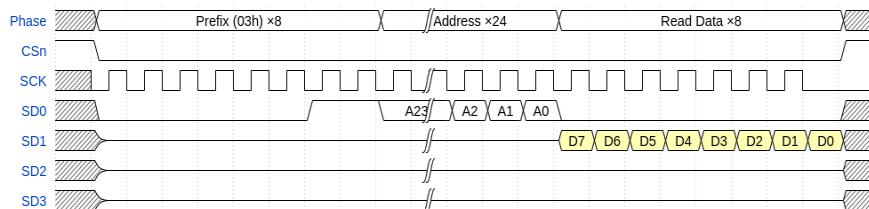


图130展示了 03h 串行读取命令。本节涉及一些QSPI闪存/SRAM/PSRAM设备常用的QSPI读写命令；具体命令详情请参阅对应的QSPI设备数据手册。例如，Winbond发布的W25Q16JV数据手册包含本节所述全部读取命令的说明。

应用前述提出的五阶段结构，03h QSPI事务分解如下：

1. 8位前缀，串行宽度下（前缀= 0x03）
2. 24位地址，串行宽度下
3. 无后缀（长度为0）
4. 无虚拟位（长度为0）
5. 数据位，串行宽度下

所有QMI访问的地址位数固定为24位。数据位数取决于传输大小：本图示显示传输8位数据，代表处理器读取的未缓存字节。

M0\_RFMT/M1\_RFMT寄存器配置数据阶段使用的所有其他参数，例如串行接口宽度。

QSPI总线由四条数据线 SD3至 SD0组成。在串行宽度下，主机通过 SD0线输出数据，设备则通过相反方向的 SD1线返回数据。在序列 SPI和双 SPI宽度的传输阶段，SD3和 SD2处于未驱动状态，通常被上拉。数据位 D7至 D0后的阴影背景表示传输方向为设备到主机。更高的接口宽度使用 SDx线路进行双向通信。

图131。0Bh读取命令在地址与数据之间加入了8个空周期，以支持更高的总线频率。

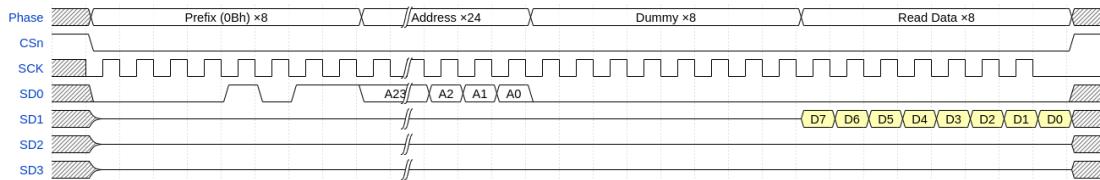


图131显示了0Bh串行读取命令，这是03h的一种常见变体。0Bh命令在地址与数据阶段之间插入空周期，有助于掩盖QSPI设备内存阵列的初始访问延迟。这使得更高的工作频率成为可能。

根据之前介绍的五阶段结构，0Bh QSPI 事务具体分解如下：

1. 8位前缀，串行宽度（前缀 = 0x0b）
2. 24位地址，串行宽度下
3. 无后缀（长度为0）
4. 八位虚拟位，按串行宽度处理
5. 数据位，串行宽度下

在串行宽度下，QMI在虚拟阶段持续将 SD0线拉低，因该线在此宽度下为单向。在双SPI和四SPI宽度下，SD0在虚拟阶段与SD1至 SD3一起进入三态状态。

QMI在传输间隙保持时钟线为低电平，预期数据在每个时钟脉冲的上升沿被采集。按照传统摩托罗拉SPI术语，时钟极性为0，时钟相位为0。不支持其他时钟极性和相位设置。为确保数据在上升沿稳定，新数据在每个下降沿发出。

当禁用传输链（参见第12.14.2.1节）时，QMI利用此时钟特性，在读取过程中抑制最后的时钟脉冲。此举通过避免不必要的SCK切换，并避免无意中请求紧随所请求数据之后的数据，从而节省能量。QMI仍保留一个完整的SCK周期，在该周期内最后一个数据有效，并且仍在 SCK上升沿即将触发的时点捕获数据（参见第12.14.3节），但实际的SCK时钟脉冲被抑制。

图132。一条EBh四路I/O读命令。命令前缀为串行传输，但地址和数据每周期为4位。

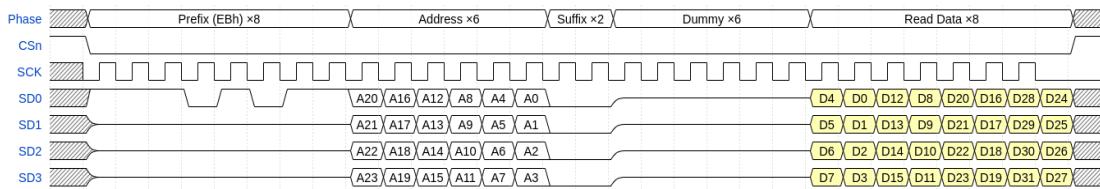


图132显示了四宽度读取传输。在本例中，命令前缀仍以串行宽度传输，但随后采用完整四宽度，因为前缀用于标识访问宽度。

应用先前介绍的五阶段结构，QSPI事务拆分如下：

1. 8位前缀，串行宽度（前缀 = 0xeb）
2. 24位地址，四宽度
3. 8位后缀，四宽度（后缀 = 0x00）
4. 24位虚拟位，四宽度
5. 数据位，四倍宽度

后缀是命令前缀的扩展，置于地址位之后，以避免扩展初始访问。

延迟。用于前缀和后缀的位模式通过M0\_RCMD/M1\_RCMD寄存器（用于读取）和M0\_WCMD/M1\_WCMD寄存器（用于写入）进行配置。EBh四倍I/O读取命令中后缀的一个常见用途是进入所谓的连续读取模式，其中跳过下一条命令的前缀（假定与当前命令相同），以减少下一次读取访问所需的周期数。

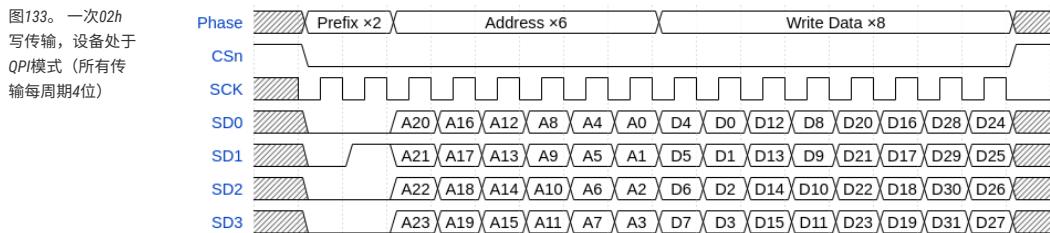


图133展示了四倍宽度的写命令。在此示例中，命令前缀也以四倍模式发出，这在QSPI RAM中较为常见。由于读写命令可自由混合，丢弃前缀（如闪存连续读取模式）实用性较低，因此QSPI RAM设备通常支持一个QPI模式，在该模式下命令前缀同样以四倍宽度发出，以降低每次访问的开销。

应用先前介绍的五阶段结构，QSPI事务拆分如下：

1. 8位前缀，四倍宽度 (prefix = **0x02**)
2. 24位地址，四宽度
3. 无后缀 (0位)
4. 无虚位
5. 数据位，四倍宽度

需要注意图中展示的位序和字节序。SPI通常在每个字节内采用最高有效位优先传输。当每个周期传输多个位（使用 SD0、SD1、SD2 和 SD3 数据线并行时），编号较高的数据线传输更高位的比特。图133中数据传输的第一个周期传输了第一个字节的四个最高有效位。最高有效位（第7位）在 SD3 上传输，而这些位中最低有效的（第4位）在 SD0 上传输。

由于RP2350采用小端模式，更高字节地址对应更高数值权重。图133展示了从主机在地址阶段发送的初始地址开始，跨越四个连续字节地址的32位值传输。数据阶段的前两个周期传输第一个字节，该字节包含32位值的最低8位。数据阶段的最后两个周期传输最后一个字节，该字节包含32位值的最高8位（包括第31位至第24位）。

#### 12.14.2.1. 传输链

参考图132，该图显示带有EBh串行前缀的32位QSPI读取，可见发出前缀和地址（14个周期）及等待初始读取延迟（额外8个周期）所花费的时间，超过了实际传输数据（8个周期）的时间。该开销仅留极少部分理论最大QSPI吞吐量用于从闪存传输数据，从而限制了直接代码执行的性能。

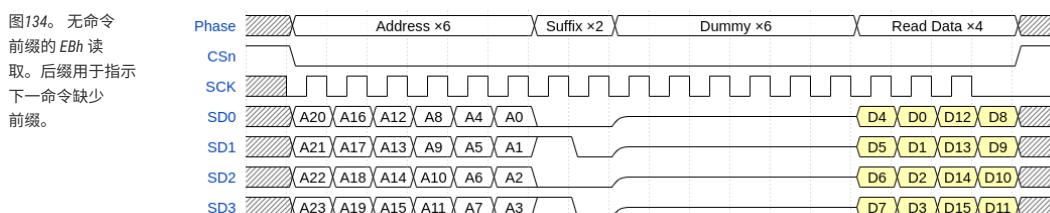
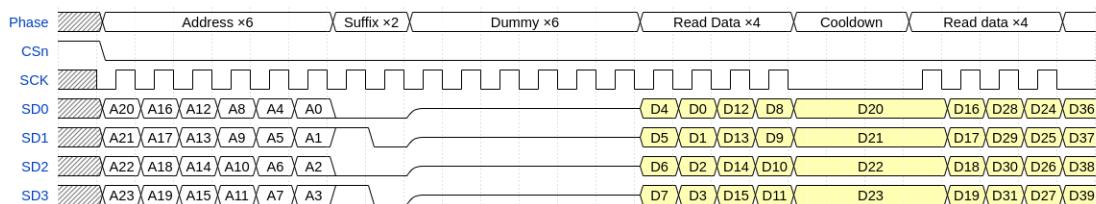


图134展示了如何通过连续读取模式来改进此过程，该模式使用后缀（此处为 **0xa0**）表示下一条命令缺少命令前缀。此示例仅传输16位数据（例如处理器执行的未缓存半字读取）。后缀的传输实际上是免费的，因为它们在从地址发出至QSPI设备内部存储返回首批数据的延迟等待期间传输。然而，这仍导致QSPI总线的大多数周期用于发出地址和等待，而非传输数据。

因此，QSPI存储的随机访问性能远低于其顺序访问性能。

图135。一次EBh  
读取，随  
后顺序读取链至下  
一次传输。



QMI的传输链功能利用了顺序访问与非顺序访问速度的差异。图135展示了两个顺序地址的半字读取（即第二次传输地址为第一次传输地址加二），且M0\_TIMING.COOLDOWN/M1\_TIMING.COOLDOWN设置为非零值。

在图134中，QMI抑制了最后一个时钟脉冲，并在最后一笔数据传输后立即释放芯片选择信号。当启用传输链式处理时，如图135所示，QMI不会抑制最后一个时钟脉冲，而是保持芯片选择信号保持有效。芯片选择信号将保持该状态一段时间，由COOLDOWN寄存器字段配置，等待下一次传输。随后，QMI通过向当前传输附加更多时钟来执行下一笔传输。

芯片选择信号在整个过程中均保持有效，而非在命令之间释放再重新拉低。要利用传输链式处理，下一笔传输必须满足以下条件：

- 传输方向与上一笔相同（读/写）
- 地址连续于上一笔传输（等于前一个地址加上前一次传输的大小）
- 地址位于与上一笔传输相同的窗口内（相同芯片选择信号）
- 上一笔传输未到达页边界中断（由M0\_TIMING.PAGEBREAK或M1\_TIMING.PAGEBREAK配置）

这显著提升了长时间未缓存的线性传输吞吐量，例如使用XIP流外设（第4.4.3节）或执行冷代码序列，而这些序列通常会连续多次未命中缓存。

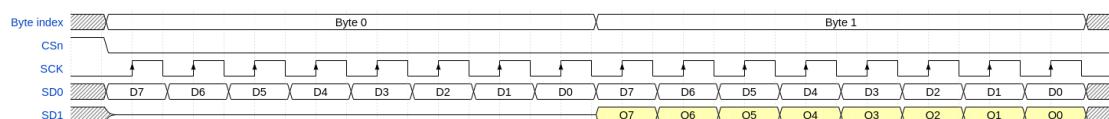
该过程可持续进行任意次数的传输。通过单地址传输链，能够读取典型闪存设备的完整内容。

请注意，传输链功能可能会略微降低随机访问性能。若下一次传输非连续，则必须释放片选信号，并可能需在片选信号保持高电平的一定最短时间内停留（视QSPI设备的时序要求而定），然后重新断言片选以发出新地址。若不使用传输链，片选信号将在前一次传输结束后立即释放，以避免部分延时。可通过调整COOLDOWN定时器寄存器参数予以缓解，以避免片选信号长时间保持断言状态，因连续传输通常时间间隔较短。

### 12.14.3. 时序

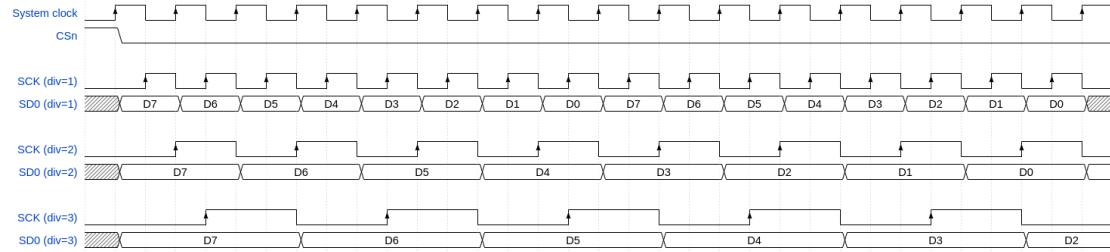
QMI运行于SPI模式0，在每个SCK上升沿捕获数据。新数据在每个随后的下降沿被断言。如图136所示，第一个输出数据与片选信号的断言同时启动。

图136。QM  
I使用的双向S  
PI传输。



QMI的时序以系统时钟为参考。由于系统时钟相对于外部信号通常较快，M0\_TIMING.CLKDIV/M1\_TIMING.CLKDIV字段可将SCK和数据信号统一按整数倍数减速。

图137。 CLKD  
IV 控制设置每个  
内存窗口中每个 SC  
K 周期所对应的系  
统时钟周期  
数。



QMI 采用 DDR 输入/输出寄存器，实现输出信号生成与输入采样的半系统时钟周期分辨率。此设计使 QMI 支持奇数分频，包括除以一（即 SCK 频率等同于系统时钟频率）。

### ① 注意

在实际应用中，最大 SCK 频率受限于连接的 QSPI 设备性能、PCB 布局的信号完整性以及焊盘中的 IO 延迟。详见第 12.14.3.4 节。

#### 12.14.3.1 输入采样与 RXDELAY

QMI 于 SCK 上升沿采样输入数据（参见第 12.14.3 节）。当往返延迟超过半个 SCK 周期时，为增加输入延迟寄存器的延迟，可使用 M0\_TIMING.RXDELAY / M1\_TIMING.RXDELAY。RXDELAY 的延迟以半个系统时钟周期为单位，而非 SCK 周期。

#### 12.14.3.2 片选时序

为节约能耗，事务完成后片选信号会被取消断言。若需在事务完成后保持片选信号断言状态，请使用 M0\_TIMING.COOLDOWN / M1\_TIMING.COOLDOWN。此操作可降低延迟并提升总线吞吐量。

芯片选择信号可通过 M0\_TIMING.SELECT\_SETUP/M1\_TIMING.SELECT\_SETUP 提前一个系统时钟周期断言。某些闪存设备在非常高的 SCK 频率下需要此项设置。若未设置此项，QMI 会在第一个 SCK 上升沿之前提前半个 SCK 周期断言芯片选择信号。此操作与第一笔数据在 SDx 上的断言同时发生。

芯片选择保持时间也可通过 M0\_TIMING.SELECT\_HOLD/M1\_TIMING.SELECT\_HOLD 额外延长最多 3 个系统时钟周期。

为强制芯片选择信号保持断言的最长时间，应使用 M0\_TIMING.MAX\_SELECT/M1\_TIMING.MAX\_SELECT。此设置对 PSRAM 设备尤为重要，因为其在取消选择时必须执行内部 DRAM 刷新周期。

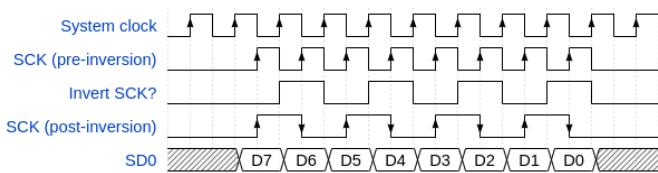
为强制芯片选择信号保持非断言的最短时间，应使用 M0\_TIMING.MIN\_DESELECT/M1\_TIMING.MIN\_DESELECT。

#### 12.14.3.3. 双倍传输速率 (DTR)

某些 QSPI 存储器设备在 SCK 的双边沿传输数据。该功能称为 **double transfer rate** (DTR)，在特定数据传输速率下允许降低 SCK 频率，从而减少电磁辐射和切换外部时钟的能耗。要启用 DTR 模式（针对窗口和方向），请设置 M0\_RFMT.DTR/M1\_RFMT.DTR 标志（读取）或 M0\_WFMT.DTR/M1\_WFMT.DTR 标志（写入）。

QMI 通过将时钟频率减半，同时保持数据速率，实现 DTR。为实现此目的，QMI 反转交替的单传输速率 SCK 时钟周期，将低-高-低-高序列转换为低-高-高-低序列。当禁用 DTR 时，QMI 于 SCK 下降沿发送数据，并于上升沿采样。启用 DTR 时，QMI 在两 SCK 边沿之间的中点发送数据，并在每个边沿采样，如图 138 所示。

图138。 DTR通过将SCK频率减半，同时保持数据传输速率不变来实现。



启用DTR模式不会改变数据时序，仅修正 SCK的时序。数据在本应出现 SCK负边沿的位置被传输，假设时钟频率未减半。

启用DTR时，传输的前缀和虚拟阶段仍保持单倍传输速率。在这些阶段，数据位被加倍以匹配减半的 SCK频率，确保每个上升沿都有新数据准备就绪。图139显示第一个字节（命令前缀）为单倍传输速率，第二个字节（地址和数据）为双倍传输速率。

图139。部分启用DTR的传输仍然采用单倍传输速率：实际上每个数据位被发送两次。

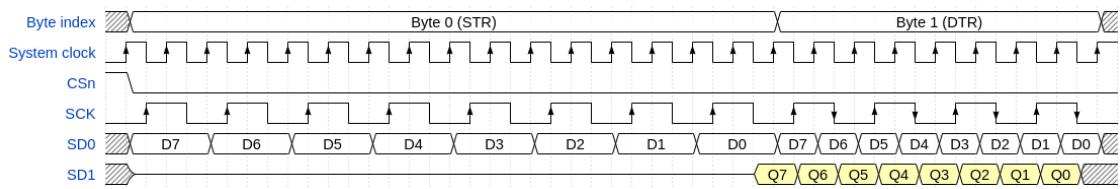


图139中 SCK线上的箭头表示 SCK的有效边沿（数据采样点）。访问的单传输速率部分预期在上升沿捕获数据。访问的双传输速率部分预期在两个沿上捕获数据。

从设备到主机传输的数据同样在 SCK的两个沿上启动。每当QMI启动新的时钟沿时，信号传递过程中存在延迟，包括RP2350引脚输出延迟、QSPI设备 SCK至SDx的延迟，以及返回RP2350 SDx引脚输入的延迟。QMI在启动下一SCK沿时同时捕获数据，加上由M0\_TIMING.RXDELAY/M1\_TIMING.RXDELAY配置的延迟。由 SCK输出返回 SDx输入的往返延迟提供了 SDx输入保持时间。如果输入建立时间不足，可以增加 RXDELAY。更多信息请参阅特定QSPI设备的数据手册及第12.14.3.4节。

#### 12.14.3.4. AC 时序参数

QMI 接口采用内部系统时钟进行时序控制。不同 QMI 引脚的输入或输出时钟偏斜保持在最小范围内。如其他章节所述，通过使用额外的时钟周期延迟支持任何额外的建立时间或保持时间。时钟偏斜值取决于仅考虑专用 QSPI 引脚 (**QSPI\_SS**、**QSPI\_SD[3:0]**、**QSPI\_SCLK**) 还是包括 Bank 0 GPIO XIP 特殊功能（用于额外的 QMI 片选）。不同封装选项的时钟偏斜时序不同，如下所示。

表 1291. QMI 时序偏斜

接口	典型偏斜 (ps)	最大偏斜 (ps)
QSPI 输入	15	25
QSPI 输出	100	180
Bank 0 GPIO (QFN-60) 输出	1080	1725
Bank 0 GPIO (QFN-80) 输出	1280	2100

了解从运行于系统时钟的内部寄存器到输出引脚的延迟，以及从输入引脚到运行于系统时钟的采样寄存器的延迟，亦十分重要。表1292列出了QSPI输入和输出以及GPIO输出在工艺、电压和温度最坏情况下的时序。请注意，该延迟会根据表中显示的 VDDIO电压等级而变化。

表 1292. QMI 时序延迟

路径	最大延迟 (纳秒) VDDIO=3.3V	最大延迟 (纳秒) VDDIO=1.8V
QSPI输入到系统时钟	1.5	1.2
系统时钟到QSPI输出	2.5	3.6

路径	最大延迟 (纳秒) VDDIO=3.3V	最大延迟 (纳秒) VDDIO=1.8V
系统时钟到GPIO (QFN-60) 输出	3.5	4.9
系统时钟到GPIO (QFN-80) 输出	4.1	5.4

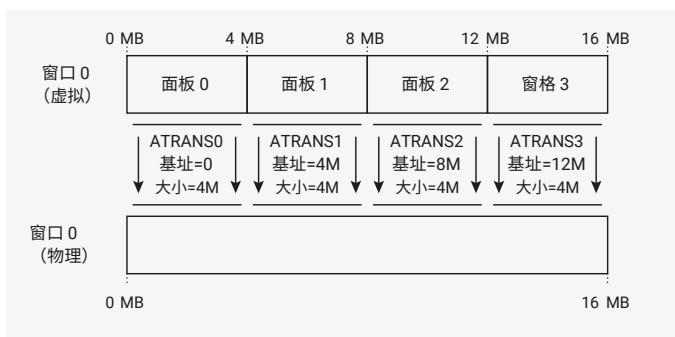
#### 12.14.4. 地址转换

QMI针对处理器或DMA请求的虚拟地址应用了可配置映射，转换为传输到外部QSPI设备的物理地址。此操作针对每个16 MB芯片选择窗口单独执行。禁止在不同设备之间映射内容。

每个窗口划分为四个窗格，每个窗格独立映射到该窗口对应的物理地址空间。

如图140所示，QMI复位时的默认配置为虚拟地址与物理地址之间的1:1同一映射。在该状态下，地址映射无效，外部QSP I设备的整个16 MB地址空间被直接映射到系统地址空间。

图140。默认情况下，每个窗口均设置为将完整的16 MB虚拟地址空间直接1:1映射到16 MB物理地址空间。



每个窗格对应该窗口的四个ATRANSx寄存器之一：窗口0的ATRANS0至ATRANS3，窗口1的ATRANS4至ATRANS7。

每个窗格的虚拟基址固定，以4 MB为单位递增分配。映射该窗格到物理地址空间有两个可配置参数：

- **BASE**：定义虚拟地址窗格中偏移地址0对应的物理地址。以4 kB（一个闪存扇区）为单位配置，范围从0至（16 MB减4 kB）。
- **SIZE**：定义该窗格所映射的地址空间大小。以4 kB（一个闪存扇区）为单位配置，范围从0至4 MB。

映射从窗格的起始位置开始。一个SIZE为1 MB，将该窗格虚拟地址范围的前1 MB映射到下游内存，其余部分未映射。一个SIZE为0表示该虚拟地址窗格内没有地址可被访问。访问超出当前配置的SIZE将返回总线错误，且不会传递至下游QSPI总线。因此，这些访问不会对外部存储设备产生任何影响。

图141。窗格的BASE定义其物理映射的起始位置。SIZE定义映射的延伸范围。SIZE为0表示该窗格未映射任何地址。

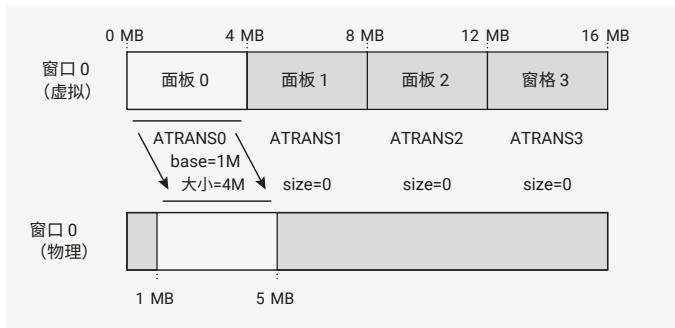


图141显示了一个示例映射，其中芯片选择0的虚拟地址空间前4 MB（虚拟地址偏移 0x000000 至 0x3fffff 包含）映射到起始于1 MB偏移的4 MB物理地址窗口（物理地址偏移 0x100000 至 0x4fffff 包含）。此映射可用于包含1 MB

引导加载程序应用程序，随后是4 MB用户应用程序的闪存。理想情况下，用户应用程序不应了解由引导加载程序定义的闪存布局；如此，相同的应用程序可在不同引导加载程序实现下运行。

虚拟地址到物理地址的映射通过使用户应用程序的存储位置（从1 MB开始）与其在系统地址空间中出现的地址（从0 MB开始）相互独立，解决了该问题。

#### 12.14.4.1 Bootrom对地址转换的支持

Bootrom能在启动时自动配置地址转换，使存储在物理闪存任意位置的二进制文件，在运行时表现为起始于0地址的闪存地址。

当启动映像位于闪存分区（第5.1.2节）时，该过程自动完成；且可依据启动可执行文件的 **IMAGE\_DEF** 中指定的滚动窗口偏移量进行手动调整（第5.1.4节）。

引导只读存储器源代码及其文档通常将QMI **ATRANS** 映射称为“滚动窗口”，这是由于16 MB边界上的模地址回绕——详见第5.1.19节。

#### 12.14.4.2. 翻译与XIP缓存

QMI地址翻译在系统XIP缓存（第4.4.1节）之后进行。因此，对于该翻译而言，XIP缓存是一个虚拟缓存，因为QMI内部执行的地址翻译对XIP缓存是透明的。

因此，对QMI地址翻译的更改须刷新XIP缓存。从缓存视角来看，翻译更改导致QMI内存内容在缓存的下游地址空间内以与缓存内容不一致的方式移动，故须刷新以恢复一致性。至少，任何其 **ATRANSx** 寄存器（**ATRANS0**至**ATRANS7**）被修改，且可能以干净或脏状态缓存的虚拟地址，必须刷新。最简单的方法可能是清空整个缓存。

QMI 的地址映射引发另一种风险：同一物理地址可能映射至多个虚拟地址，因而可能在 XIP 缓存中被多次分配。当通过缓存的虚拟地址别名写入物理地址时，XIP 缓存不会将改动传播至其他别名。为避免此类问题，请勿在同一时间允许同一可写物理地址存在多个别名。对只读存储器进行别名映射通常是安全的。

存在于不同时点的别名（例如跨越 RTOS 上下文切换边界时）可通过在地址映射变更时进行适当清理与刷新以保持一致性。

#### 12.14.5. 直接模式

在直接模式下，AHB XIP 地址窗口与 QSPI 总线断开连接，且总线通过 TX/RX FIFO 对进行控制，类似于普通 SPI 外设。在此状态下，XIP 窗口不可访问。尝试访问该窗口将导致总线故障。该模式用于对 QSPI 总线的底层访问，例如发出闪存擦除/编程命令，或访问 QSPI 设备状态寄存器时。

**所有直接模式操作均通过**`DIRECT_CSR`**控制，数据通过**`DIRECT_TX`**和**`DIRECT_RX`**进行交换。启用直接模式时，须先设置**`DIRECT_CSR.EN`**，然后轮询**`DIRECT_CSR.BUSY`**至低电平，以确保在启用直接模式时任何正在进行的 XIP 传输已完成。**

直接模式拥有独立的时钟分频器和RX采样延迟，分别由`DIRECT_CSR.CLKDIV`和`DIRECT_CSR.RXDELAY`配置。此设置独立于**M0\_TIMING/M1\_TIMING**中配置的每窗口参数，原因在于用于控制的串行命令的频率限制可能与用于执行就地访问的数据访问频率不同。

每次向`DIRECT_TX`推送数据时，QMI将向QSPI总线发送8位或16位FIFO数据。可选地，同等位数的数据会同步采样并返回至`DIRECT_RX`。时钟初始为低电平，数据始终在 **SCK**上升沿被采集，并在随后的下降沿完成转换。

将数据推送至 `DIRECT_TX` 后，`DIRECT_CSR.BUSY` 将变为高电平，并保持高电平，直到所有直接模式操作完成。即使未返回任何 RX 数据，该方法依然有效，因此比轮询 RX FIFO 状态更可靠。**BUSY** 标志

传输完成后，BUSY标志保持高电平，持续半个SCK周期，以确保当其用于驱动片选信号时的安全时序——详见第 12.14.5.2 节。

QMI 不会向满的 RX FIFO 推送数据，也不会因 FIFO 满而丢弃数据——接口将暂停，直到系统弹出 DIRECT\_RX。这避免了处理器在直接模式操作中频繁中断时导致 RX 数据丢失的常见问题，但软件必须注意，无论何时轮询 DIRECT\_CSR.BUSY 变为低电平，都应同时检查 RX FIFO，否则 FIFO 满时可能会导致死锁。

### 12.14.5.1. DIRECT\_TX 中的控制

TX FIFO 承载控制信息及数据，数据位于最低 16 位，控制信息位于紧挨其上的更高位：

- DIRECT\_TX.NOPUSH 抑制匹配该 TX 数据的 DIRECT\_RX 推送。此操作避免在传输起始阶段推送控制或地址信息时产生无效数据。
- DIRECT\_TX.DWIDTH 表示该 FIFO 记录的数据宽度。数值 0 表示最低有效的 8 位包含数据，数值 1 表示最低有效的 16 位包含数据。该参数亦决定匹配的 DIRECT\_RX 条目所返回的数据量。
- DIRECT\_TX.IWIDTH 为用于时钟输出该 FIFO 记录的接口宽度（单路/双路/四路）。对应的 RX 数据以相同宽度进行采样。
- DIRECT\_TX.OE 控制双向传输中引脚的方向。对于串行 IWIDTH，该设置被忽略，因为 SD0 始终为输出，而 SD1 始终为输入。在双路或四路宽度模式下，必须设置该位以启用本 FIFO 记录持续期间的输出驱动。当 IWIDTH 为双路或四路且 OE 未设置时，TX 数据处于无关状态。

当所有控制位均为零时，默认进行8位串行传输，返回采样的8位数据。因此，您可以忽略控制位，将其视为纯粹的8位数据FIFO。

### 12.14.5.2. 芯片选择控制

通过DIRECT\_CSR，有两种芯片选择驱动方式：

- 设置DIRECT\_CSR ASSERT\_CS0N 和 DIRECT\_CSR ASSERT\_CS1N 时，将立即驱动对应的芯片选择信号拉低。
- DIRECT\_CSR.AUTO\_CS0N 和 DIRECT\_CSR.AUTO\_CS1N 配置对应的芯片选择信号在接口忙碌时（即因先前 DIRECT\_TX 推送导致 DIRECT\_CSR.BUSY 标志为高电平时）被拉低。

#### 重要

ASSERT\_CSxN 字段会无条件断言芯片选择信号，包括在 DIRECT\_CSR.EN 清除时。软件必须避免在可能存在 XIP 传输的情况下设置这些字段。

### 12.14.6. 寄存器列表

QMI 控制寄存器起始地址为 `0x400d0000`，在 SDK 中定义为 `XIP_QMI_BASE`。

表1293。QMI 寄存器列表

偏移量	名称	说明
0x00	DIRECT_CSR	直接串行模式的控制与状态 直接串行模式允许处理器发送和接收原始串行帧，用于外部存储器设备的编程、配置及控制。仅支持 SPI 模式 0 (CPOL=0, CPH A=0)。
0x04	DIRECT_TX	直接模式的发送 FIFO

偏移量	名称	说明
0x08	DIRECT_RX	直接模式的接收FIFO
0x0c	M0_TIMING	内存地址窗口0的时序配置寄存器
0x10	M0_RFMT	内存地址窗口的读取传输格式配置0.
0x14	M0_RCMD	用于从内存地址读取的命令常量窗口0
0x18	M0_WFMT	内存地址窗口0的写入传输格式配置
0x1c	M0_WCMD	用于写入内存地址窗口的命令常量0.
0x20	M1_TIMING	内存地址窗口1的时序配置寄存器
0x24	M1_RFMT	内存地址窗口的读取传输格式配置1.
0x28	M1_RCMD	用于从内存地址读取的命令常量窗口1
0x2c	M1_WFMT	内存地址窗口1的写传输格式配置
0x30	M1_WCMD	用于写入内存地址窗口的命令常量1.
0x34	ATRANS0	配置XIP虚拟地址的地址转换 0x000000至0x3FFFF (起始于+0 MiB的4 MiB窗口)
0x38	ATRANS1	配置XIP虚拟地址的地址转换 0x400000至0x7FFFF (起始于+4 MiB的4 MiB窗口)
0x3c	ATRANS2	配置XIP虚拟地址的地址转换 0x800000 到 0xbffff (从 +8 MiB 起始的 4 MiB 窗口)。
0x40	ATRANS3	配置XIP虚拟地址的地址转换 0xc00000 到 0xfffffff (从 +12 MiB 起始的 4 MiB 窗口)。
0x44	ATRANS4	配置XIP虚拟地址的地址转换 0x1000000 到 0x13ffff (从 +16 MiB 起始的 4 MiB 窗口)。
0x48	ATRANS5	配置XIP虚拟地址的地址转换 0x1400000 到 0x17ffff (从 +20 MiB 起始的 4 MiB 窗口)。
0x4c	ATRANS6	配置XIP虚拟地址的地址转换 0x1800000 到 0x1bffff (从 +24 MiB 起始的 4 MiB 窗口)。
0x50	ATRANS7	配置XIP虚拟地址的地址转换 0x1c00000 到 0x1fffff (从 +28 MiB 起始的 4 MiB 窗口)。

## QMI: DIRECT\_CSR 寄存器

偏移: 0x00

**描述**

直接串行模式的控制与状态

直接串行模式允许处理器发送和接收原始串行帧，用于外部存储器设备的编程、配置及控制。仅支持SPI模式0（CPOL=0，CPHA=0）。

表 1294。  
DIRECT\_CSR 寄存器

位	描述	类型	复位
31:30	<b>RXDELAY:</b> 以系统时钟周期一半为单位，延迟读取数据采样时序。（不一定 是SCK周期的一半。）	读写	0x0
29:22	<b>CLKDIV:</b> 直接串行模式的时钟分频系数。除数1至255直接编码，最大除数256用CLKDIV=0表示。  时钟分频系数可由软件动态修改，无需停止或协调串行接口。串行接口在开始传输新字节时采样最新时钟分频值。	读写	0x06
21	保留。	-	-
20:18	<b>RXLEVEL:</b> 当前DIRECT_RX FIFO的级别。	只读	0x0
17	<b>RXFULL:</b> 当值为1时，DIRECT_RX FIFO已满。串行接口将暂停，直到数据被弹出；当DIRECT_TX FIFO为空或DIRECT_RX FIFO满时，接口不会开始新的串行帧。	只读	0x0
16	<b>RXEMPTY:</b> 当值为1时，DIRECT_RX FIFO为空。若处理器尝试读取更多数据，FIFO状态保持不变，但返回值为未定义。	只读	0x0
15	保留。	-	-
14:12	<b>TXLEVEL:</b> DIRECT_TX FIFO 当前级别	只读	0x0
11	<b>TXEMPTY:</b> 当为1时，DIRECT_TX FIFO 当前为空。除非处理器继续推送数据，否则传输将在当前8位串行帧结束后停止，且BUSY信号将置低。	只读	0x0
10	<b>TXFULL:</b> 当为1时，DIRECT_TX FIFO 当前已满。如果处理器尝试写入更多数据，该数据将被忽略。	只读	0x0
9:8	保留。	-	-
7	<b>AUTO_CS1N:</b> 当为1时，每当BUSY标志被置位时，自动断言CS1n芯片选择线。	读写	0x0
6	<b>AUTO_CS0N:</b> 当为1时，每当BUSY标志被置位时，自动断言CS0n芯片选择线。	读写	0x0
5:4	保留。	-	-
3	<b>ASSERT_CS1N:</b> 当为1时，断言（即拉低）CS1n芯片选择线。  请注意，此规则即使在DIRECT_CSR_EN为0时亦适用。	读写	0x0
2	<b>ASSERT_CS0N:</b> 当为1时，断言（即拉低）CS0n芯片选择线。  请注意，此规则即使在DIRECT_CSR_EN为0时亦适用。	读写	0x0

位	描述	类型	复位
1	<p><b>BUSY:</b> 直接模式忙标志。若为1，则表示数据当前正在移入或移出（或者若接口未因RX FIFO阻塞则会进行移入/移出），芯片选择信号不得取消断言。</p> <p>若启用直接模式且内存映射传输仍在进行，忙标志亦将被置为1。直接模式阻止新的内存映射传输，但无法中止已在进行的传输。若可能存在内存映射传输正在进行，应在断言芯片选择信号前轮询忙标志至0。</p> <p>（实际上，您通常会通过其他手段发现该时序条件，因为启用直接模式后任何后续的内存映射传输都会返回总线错误，而此类错误难以忽视。）</p>	只读	0x0
0	<p><b>EN:</b> 使能直接模式。</p> <p>在直接模式下，软件控制芯片选择线，可通过向DIRECT_TX FIFO推送数据，并从DIRECT_RX FIFO弹出等量数据来执行直接SPI传输。</p> <p>启用直接串行模式时，内存映射访问将引发总线错误。</p>	读写	0x0

## QMI: DIRECT\_TX 寄存器

偏移: 0x04

### 描述

直接模式的发送FIFO

表 1295。  
DIRECT\_TX 寄存器

位	描述	类型	复位
31:21	保留。	-	-
20	<p><b>NOPUSH:</b> 禁止对应此TX FIFO条目的RX FIFO推送操作。</p> <p>用于避免在SPI传输开始时推送命令时，RX FIFO中出现无效数据。</p>	WF	0x0
19	<p><b>OE:</b> 输出使能（高电平有效）。对于单宽度（SPI），此字段被忽略，SD0始终设置为输出，SD1始终设置为输入。</p> <p>对于双宽和四宽（DSPI/QSPI），该字段设置传输此FIFO记录时相关SDx引脚是否设为输出。在此情况下，命令/地址应设置OE，数据传输则应根据传输方向设置或清除OE。</p>	WF	0x0
18	<b>DWIDTH:</b> 数据宽度。若为0，硬件传输DIRECT_TX DATA字段的8位最低有效位，并在DIRECT_RX中的8位最低有效位返回8位数据；若为1，使用完整16位宽度。8位和16位传输可自由混合。	WF	0x0
17:16	<b>IWIDTH:</b> 配置该FIFO记录是否以单、双或四接口宽度（0/1/2）传输。不同宽度可自由混合使用。	WF	0x0
	枚举值：		
	0x0 → S: 单宽度		

位	描述	类型	复位
	0x1 → D：双宽度		
	0x2 → Q：四宽度		
15:0	<p><b>DATA：</b> 推入此处的数据将在SCK下降沿（若为首次脉冲，则在SCK第一个上升沿之前）时钟输出。对于每个输出的字节，接口将在SCK上升沿同时采样一个字节，并将其推送至DIRECT_RX FIFO。</p> <p>对于16位数据，先传输最低有效字节。</p>	WF	0x0000

## QMI：DIRECT\_RX 寄存器

偏移：0x08

### 说明

直接模式的接收FIFO

表 1296。  
DIRECT\_RX 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p>串行接口每输出一个字节的同时，将同步时钟输入一个字节，该字节存入此FIFO。当FIFO满时，串行接口将暂停，以避免数据丢失。</p> <p>当16位数据写入TX FIFO时，相应的RX FIFO中也将写入16位数据。最低有效字节是第一个接收的字节。</p>	RF	0x0000

## QMI：M0\_TIMING, M1\_TIMING 寄存器

偏移量：0x0c, 0x20

### 描述

内存地址窗口 0/1 的时序配置寄存器。

表 1297。  
M0\_TIMING,  
M1\_TIMING 寄存器

位	描述	类型	复位
31:30	<p>冷却周期：芯片选择冷却时长。内存传输完成后，芯片选择保持断言状态，持续 <math>64 \times</math> 冷却周期的系统时钟周期，加半个 SCK 时钟周期（奇数 SCK 除数时向上取整）。冷却周期结束后，芯片选择将始终撤销，以节省功耗。</p> <p>若下一次内存访问在冷却周期内到达，QMI 可能在当前进行的 SPI 传输中追加更多 SCK 周期，而非开启新传输。此举可降低访问延迟并提升总线吞吐量。</p> <p>具体要求下一次访问需保持相同方向（读/写）、访问相同内存窗口（片选 0/1），且地址需紧接上一次传输的顺序。若任一条件不满足，新访问将先解除片选信号，然后启动新传输。</p> <p>若 COOLDOWN 为 0，或已达到 PAGEBREAK 配置的地址对齐，或已达到片选最大断言限制 MAX_SELECT，则跳过冷却期，片选信号将在传输结束后一半 SCK 周期内始终解除断言。</p>	读写	0x1

位	描述	类型	复位
29:28	<p><b>PAGEBREAK:</b> 启用分页断点后，跨越特定2的幂对齐地址边界时，片选信号将自动解除断言。即使下一次访问地址在分页边界前与上次访问地址连续，下一次访问也将始终启动新的读/写SPI突发传输。</p> <p>部分闪存与PSRAM设备禁止单次读/写传输跨越分页边界，或对跨页传输的工作频率加以限制。此选项允许QMI安全地支持这些设备。</p> <p>当禁用COOLDOWN时，此字段无效。</p>	读写	0x0
	枚举值：		
	0x0 → NONE：不强制页面边界		
	0x1 → 256：在跨越256字节页面边界时终止突发		
	0x2 → 1024：在跨越1024字节四页边界时终止突发		
	0x3 → 4096：在跨越4096字节扇区边界时终止突发		
27:26	保留。	-	-
25	<p><b>SELECT_SETUP:</b> 在芯片选择信号断言与SCK第一个上升沿之间增加最多一个系统时钟周期的设置时间。</p> <p>默认设置时间为半个SCK周期，通常足够。 但对于某些高速SCK频率下的闪存设备例外。</p>	读写	0x0
24:23	<p><b>SELECT_HOLD:</b> 在SCK最后一个下降沿与该窗口芯片选择信号取消断言之间，增加最多三个系统时钟周期的保持时间。</p> <p>默认保持时间为一个系统时钟周期。请注意，闪存数据手册通常规定芯片选择有效保持时间从SCK最后一个上升沿开始计时，因此即便从最后一个下降沿计算零保持时间，也属安全。</p> <p>请注意，此为QMI保证的最小保持时间：对于低时钟分频和/或高采样延迟的读取传输，芯片选择激活保持时间可能会稍长。具体而言，若最后一个R X数据采样后两个周期的时间点晚于最后一个SCK下降沿，则保持时间应从该时间点开始计算。</p> <p>还应注意，若为节能掩盖了最终的SCK脉冲（当禁用COOLDOWN或达到PAGE_BREAK时，非DTR读取有效），QMI所有定时逻辑均视时钟脉冲依然存在。SELECT_HOLD时间自假定时钟脉冲未被掩盖时最后一个SCK下降沿出现的时间点起算。</p>	读写	0x0

位	描述	类型	复位
22:17	<p><b>MAX_SELECT:</b> 以64个系统时钟周期为单位，强制执行该窗口芯片选择的最大断言持续时间。若设为0，QMI在处理连续内存访问时（详见COOLDOWN）允许芯片选择无限期保持断言状态。</p> <p>此特性旨在满足PSRAM设备的定时约束，该设备规定最大芯片选择断言时间以便完成DRAM刷新周期。另请参见MIN_DESELECT，该参数可用以强制执行最小取消选择时间。</p> <p>若在达到MAX_SELECT时内存访问仍在进行，QMI将等待该访问完成后再取消芯片选择信号。</p> <p>计算安全的MAX_SELECT值时，必须考虑此额外时间。在最坏情况下，该过程可能为完整的串行传输，包括命令前缀和地址，数据负载最大可达一个缓存行。</p>	读写	0x00
16:12	<p><b>MIN_DESELECT:</b> 在本时间窗口芯片选择信号被取消后，芯片选择保持取消状态半个SCK周期（向上取整至系统时钟整数周期数），再加上MIN_DESELECT个系统时钟周期，QMI才会重新断言任一芯片选择引脚。</p> <p>对于需要较长最小CS取消选择时间以在取消选择期间执行内部DRAM刷新周期的PSRAM设备，可能需设置非零值。</p>	读写	0x00
11	保留。	-	-
10:8	<p><b>RXDELAY:</b> 以系统时钟周期一半为单位，延迟读取数据采样时序。（不必然为半个SCK周期。）当RXDELAY为0时，样本在SCK输出寄存器发出的SCK上升沿同时被采集至SDI输入寄存器。</p> <p>在较高的SCK频率下，可能需要增加RXDELAY以补偿焊盘的往返时延及QSPI存储器设备的时钟至Q延迟。</p>	读写	0x0
7:0	<p><b>CLKDIV:</b> 时钟分频系数。支持奇数和偶数分频系数。以1个系统时钟周期为单位定义SCK时钟周期。分频系数1至255直接编码，分频系数256则用CLKDIV=0编码。</p> <p>时钟分频系数可动态调整，即使QMI当前正在访问该地址窗口内的存储器。所有其他参数仅可在QMI空闲时更改。</p> <p>若软件为配合系统时钟频率提升而增加CLKDIV，必须在Mx_TIMING寄存器写入后，插入对任一存储窗口的虚拟访问及相应的处理器屏障/栅栏操作，以确保SCK分频变更在系统时钟切换之前生效。</p>	读写	0x04

## QMI：M0\_RFMT，M1\_RFMT 寄存器

偏移量：0x10, 0x24

### 描述

读取内存地址窗口0/1的传输格式配置。

分别配置每个传输阶段的总线宽度，并配置命令前缀、命令后缀及虚拟/转换传输阶段的长度或是否存在。仅支持24位地址。

Mx\_RFMT寄存器的复位值配置为支持基本03h串行读取传输，不需要额外配置。

表1298。  
M0\_RFMT, M1\_RFMT  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<p><b>DTR</b>：为读取命令启用双倍传输速率（DTR）：地址、后缀和读取数据阶段均在SCK的两个边沿有效。SDO数据在每个SCK边沿中心对齐启动，SDI数据在其启动后的SCK边沿捕获。</p> <p>DTR通过将时钟频率减半实现；整个传输期间，SCK的周期为<math>2 \times \text{CLK\_DIV}</math>。前缀和虚拟阶段仍为单传输速率。</p> <p>若后缀为四倍宽度，其长度必须为0或8位，以确保SCK边沿数为偶数。</p>	读写	0x0
27:19	保留。	-	-
18:16	<p><b>DUMMY_LEN</b>：命令后缀与数据阶段之间虚拟阶段的长度，单位为4位。（即四倍宽度为1个周期，双倍为2个周期，单倍为4个周期）</p> <p>枚举值：</p> <ul style="list-style-type: none"> <li>0x0 → 无：无虚拟阶段</li> <li>0x1 → 4：4个虚拟位</li> <li>0x2 → 8：8个虚拟位</li> <li>0x3 → 12：12个虚拟位</li> <li>0x4 → 16：16个虚拟位</li> <li>0x5 → 20：20个虚拟位</li> <li>0x6 → 24：24个虚拟位</li> <li>0x7 → 28：28个虚拟位</li> </ul>	读写	0x0
15:14	<p><b>SUFFIX_LEN</b>：地址后命令后缀的长度，单位为4位。（即四倍宽度为1个周期，双倍为2个周期，单倍为4个周期）</p> <p>仅支持0位和8位的数值。</p> <p>枚举值：</p> <ul style="list-style-type: none"> <li>0x0 → NONE：无后缀</li> <li>0x2 → 8：8位后缀</li> </ul>	读写	0x0
13	保留。	-	-
12	<p><b>PREFIX_LEN</b>：命令前缀长度，单位为8位（即四线宽为2个周期，双线为4个周期，单线为8个周期）</p> <p>枚举值：</p> <ul style="list-style-type: none"> <li>0x0 → NONE：无前缀</li> <li>0x1 → 8：8位前缀</li> </ul>	读写	0x1
11:10	保留。	-	-
9:8	<b>DATA_WIDTH</b> ：用于数据传输的位宽	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → S：单宽度		
	0x1 → D：双宽度		
	0x2 → Q：四宽度		
7:6	<b>DUMMY_WIDTH</b> ：虚拟周期阶段使用的位宽（如有）  若宽度为单线，则虚拟阶段期间SD0/MOSI保持低电平有效，SD1至SD3处于三态。若宽度为双线或四线，则虚拟阶段期间所有IO均处于三态。	读写	0x0
	枚举值：		
	0x0 → S：单宽度		
	0x1 → D：双宽度		
	0x2 → Q：四宽度		
5:4	<b>SUFFIX_WIDTH</b> ：地址后命令后缀使用的位宽（如有）	读写	0x0
	枚举值：		
	0x0 → S：单宽度		
	0x1 → D：双宽度		
	0x2 → Q：四宽度		
3:2	<b>ADDR_WIDTH</b> ：地址传输使用的位宽。地址阶段始终传输共计24位。	读写	0x0
	枚举值：		
	0x0 → S：单宽度		
	0x1 → D：双宽度		
	0x2 → Q：四宽度		
1:0	<b>PREFIX_WIDTH</b> ：命令前缀使用的传输宽度（如有）	读写	0x0
	枚举值：		
	0x0 → S：单宽度		
	0x1 → D：双宽度		
	0x2 → Q：四宽度		

## QMI：M0\_RCMD，M1\_RCMD寄存器

偏移量：0x14， 0x28

### 描述

用于从内存地址窗口0/1读取的命令常量。

Mx\_RCMD寄存器的复位值配置为支持基本的03h串行读取传输，无需额外配置。

表1299。  
*M0\_RCMD, M1\_RCMD*  
寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15:8	<b>SUFFIX</b> : 若Mx_RFMT_SUFFIX_LEN非零，则为地址后跟随的命令后缀位。	读写	0xa0
7:0	<b>PREFIX</b> : 若Mx_RFMT_PREFIX_LEN非零，则为每次新传输前附加的命令前缀位。	读写	0x03

## QMI: M0\_WFMT, M1\_WFMT寄存器

偏移量: 0x18, 0x2c

### 描述

内存地址窗口0/1的写传输格式配置。

分别配置每个传输阶段的总线宽度，并配置命令前缀、命令后缀及虚拟/转换传输阶段的长度或是否存在。仅支持24位地址。

Mx\_WFMT 寄存器的复位值被配置为支持基本的 02h 串行写传输。然而，必须首先通过该窗口的 XIP\_CTRL\_WRITABLE\_Mx 位启用对该窗口的写入，因为 XIP 内存默认为只读。

表 1300。  
M0\_WFMT, M1\_WFMT  
寄存器

位	描述	类型	复位
31:29	保留。	-	-
28	<b>DTR</b> : 启用写命令的双倍传输速率 (DTR)；地址、后缀和写数据阶段在 SCK 的两个边沿均处于活动状态。SDO 数据在每个 SCK 边沿中心对齐发射，SDI 数据在紧随其发射后的 SCK 边沿捕获。  DTR 通过将时钟频率减半实现；整个传输期间，SCK 的周期为 $2 \times \text{CLK\_DIV}$ 。前缀和虚拟阶段仍为单传输速率。  若后缀为四倍宽度，其长度必须为 0 或 8 位，以确保 SCK 边沿数为偶数。	读写	0x0
27:19	保留。	-	-
18:16	<b>DUMMY_LEN</b> : 命令后缀与数据阶段之间虚拟阶段的长度，单位为 4 位。（即四倍宽度为 1 个周期，双倍为 2 个周期，单倍为 4 个周期）  枚举值： 0x0 → 无：无虚拟阶段 0x1 → 4：4 个虚拟位 0x2 → 8：8 个虚拟位 0x3 → 12：12 个虚拟位 0x4 → 16：16 个虚拟位 0x5 → 20：20 个虚拟位 0x6 → 24：24 个虚拟位 0x7 → 28：28 个虚拟位	读写	0x0
15:14	<b>SUFFIX_LEN</b> : 地址后命令后缀的长度，单位为 4 位。（即四倍宽度为 1 个周期，双倍为 2 个周期，单倍为 4 个周期）  仅支持 0 位和 8 位的数值。	读写	0x0

位	描述	类型	复位
	枚举值：		
	0x0 → NONE：无后缀		
	0x2 → 8：8位后缀		
13	保留。	-	-
12	<b>PREFIX_LEN</b> : 命令前缀长度，单位为8位（即四线宽为2个周期，双线为4个周期，单线为8个周期）	读写	0x1
	枚举值：		
	0x0 → NONE：无前缀		
	0x1 → 8：8位前缀		
11:10	保留。	-	-
9:8	<b>DATA_WIDTH</b> : 用于数据传输的位宽	读写	0x0
	枚举值：		
	0x0 → S: 单宽度		
	0x1 → D: 双宽度		
	0x2 → Q: 四宽度		
7:6	<b>DUMMY_WIDTH</b> : 虚拟周期阶段使用的位宽（如有）  若宽度为单线，则虚拟阶段期间SD0/MOSI保持低电平有效，SD1至SD3处于三态。若宽度为双线或四线，则虚拟阶段期间所有IO均处于三态。	读写	0x0
	枚举值：		
	0x0 → S: 单宽度		
	0x1 → D: 双宽度		
	0x2 → Q: 四宽度		
5:4	<b>SUFFIX_WIDTH</b> : 地址后命令后缀使用的位宽（如有）	读写	0x0
	枚举值：		
	0x0 → S: 单宽度		
	0x1 → D: 双宽度		
	0x2 → Q: 四宽度		
3:2	<b>ADDR_WIDTH</b> : 地址传输使用的位宽。地址阶段始终传输共计24位。	读写	0x0
	枚举值：		
	0x0 → S: 单宽度		
	0x1 → D: 双宽度		
	0x2 → Q: 四宽度		
1:0	<b>PREFIX_WIDTH</b> : 命令前缀使用的传输宽度（如有）	读写	0x0
	枚举值：		
	0x0 → S: 单宽度		

位	描述	类型	复位
	0x1 → D：双宽度		
	0x2 → Q：四宽度		

## QMI：M0\_WCMD，M1\_WCMD 寄存器

偏移量：0x1c, 0x30

### 描述

用于对内存地址窗口 0/1 进行写入的命令常量。

Mx\_WCMD 寄存器的复位值被配置为支持基本的 02h 串行写传输，无需额外配置。

表1301。  
M0\_WCMD,  
M1\_WCMD寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:8	后缀：当Mx_WFMT_SUFFIX_LEN非零时，位于地址后的命令后缀位。	读写	0xa0
7:0	前缀：当Mx_WFMT_PREFIX_LEN非零时，每次新传输前必须添加的命令前缀位。	读写	0x02

## QMI：ATRANS0，ATRANS4寄存器

偏移：0x34, 0x44

### 描述

配置从 $n \times 4$  MiB起始的XIP虚拟地址的4 MiB窗口的地址转换。

地址转换允许程序镜像在多个物理闪存地址处原位执行（例如，用于无线升级的双缓冲闪存镜像），从而免去位置无关代码的开销。

复位时，地址转换寄存器初始化为恒等映射，因此若不需要地址转换，则可忽略。

注意，XIP缓存完全采用虚拟寻址，因此更改地址转换后必须刷新缓存。

表 1302。ATRANS0、  
ATRANS4 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26:16	<b>SIZE</b> ：此虚拟地址范围的转换孔径大小，单位为4 KiB（一个闪存扇区）。  虚拟地址的第21至12位与SIZE比较。偏移量大于SIZE时返回总线错误，且不触发QSPI访问。	读写	0x400
15:12	保留。	-	-
11:0	<b>BASE</b> ：此虚拟地址范围的物理地址基址，单位为4 KiB（一个闪存扇区）。  处理24位虚拟地址时，首先将第23至22位（最高两位）掩码为零，再将BASE加至第23至12位（高12位），以形成物理地址。转换在16 MiB边界处循环。	读写	0x000

## QMI：ATRANS1、ATRANS5 寄存器

偏移量：0x38, 0x48

**描述**

配置地址转换，适用于XIP虚拟地址0x400000至0x7FFFFF范围（从+4 MiB开始的4 MiB窗口）。

地址转换允许程序镜像在多个物理闪存地址处原位执行（例如，用于无线升级的双缓冲闪存镜像），从而免去位置无关代码的开销。

复位时，地址转换寄存器初始化为恒等映射，因此若不需要地址转换，则可忽略。

注意，XIP缓存完全采用虚拟寻址，因此更改地址转换后必须刷新缓存。

表1303。 ATRANS1,  
ATRANS5 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26:16	<b>SIZE</b> : 此虚拟地址范围的转换孔径大小，单位为4 KiB (一个闪存扇区)。  虚拟地址的第21至12位与SIZE比较。偏移量大于SIZE时返回总线错误，且不触发QSPI访问。	读写	0x400
15:12	保留。	-	-
11:0	<b>BASE</b> : 此虚拟地址范围的物理地址基址，单位为4 KiB (一个闪存扇区)。  处理24位虚拟地址时，首先将第23至22位 (最高两位) 掩码为零，再将BASE加至第23至12位 (高12位)，以形成物理地址。转换在16 MiB边界处循环。	读写	0x400

**QMI: ATRANS2, ATRANS6 寄存器**

偏移：0x3c, 0x4c

**描述**

配置地址转换，适用于XIP虚拟地址0x800000至0xbfffff范围内（起始于+8 MiB的4 MiB窗口）。

地址转换允许程序镜像在多个物理闪存地址处原位执行（例如，用于无线升级的双缓冲闪存镜像），从而免去位置无关代码的开销。

复位时，地址转换寄存器初始化为恒等映射，因此若不需要地址转换，则可忽略。

注意，XIP缓存完全采用虚拟寻址，因此更改地址转换后必须刷新缓存。

表1304。 ATRANS2,  
ATRANS6 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26:16	<b>SIZE</b> : 此虚拟地址范围的转换孔径大小，单位为4 KiB (一个闪存扇区)。  虚拟地址的第21至12位与SIZE比较。偏移量大于SIZE时返回总线错误，且不触发QSPI访问。	读写	0x400
15:12	保留。	-	-

位	描述	类型	复位
11:0	<p><b>BASE</b>: 此虚拟地址范围的物理地址基址，单位为4 KiB（一个闪存扇区）。</p> <p>处理24位虚拟地址时，首先将第23至22位（最高两位）掩码为零，再将BASE加至第23至12位（高12位），以形成物理地址。转换在16 MiB边界处循环。</p>	读写	0x800

## QMI: ATRANS3, ATRANS7 寄存器

偏移: 0x40, 0x50

### 描述

配置地址转换，适用于XIP虚拟地址0xc00000至0xfffffff范围内（起始于+12 MiB的4 MiB窗口）。

地址转换允许程序镜像在多个物理闪存地址处原位执行（例如，用于无线升级的双缓冲闪存镜像），从而免去位置无关代码的开销。

复位时，地址转换寄存器初始化为恒等映射，因此若不需要地址转换，则可忽略。

注意，XIP缓存完全采用虚拟寻址，因此更改地址转换后必须刷新缓存。

表1305。ATRANS3,  
ATRANS7 寄存器

位	描述	类型	复位
31:27	保留。	-	-
26:16	<p><b>SIZE</b>: 此虚拟地址范围的转换孔径大小，单位为4 KiB（一个闪存扇区）。</p> <p>虚拟地址的第21至12位与SIZE比较。偏移量大于SIZE时返回总线错误，且不触发QSPI访问。</p>	读写	0x400
15:12	保留。	-	-
11:0	<p><b>BASE</b>: 此虚拟地址范围的物理地址基址，单位为4 KiB（一个闪存扇区）。</p> <p>处理24位虚拟地址时，首先将第23至22位（最高两位）掩码为零，再将BASE加至第23至12位（高12位），以形成物理地址。转换在16 MiB边界处循环。</p>	读写	0xc00

## 12.15. 系统控制寄存器

这些寄存器不与任何特定外设关联。其用于控制系统级硬件或提供相关信息，例如总线结构。此处亦以软件可访问的方式提供芯片识别信息，如JEDEC IDCODE。

### 12.15.1. SYSINFO

#### 12.15.1.1. 概述

sysinfo块包含系统信息。第一个寄存器包含芯片ID，允许程序员识别芯片软件的运行版本。第二个寄存器指示所使用的

封装配置（QFN-60或QFN-80）。第三个寄存器的值始终为1。

### 12.15.1.2. 寄存器列表

sysinfo寄存器起始于基地址 `0x40000000`（在SDK中定义为SYSINFO\_BASE）。

表1306。SYSINFO寄存器列表

偏移量	名称	说明
0x00	CHIP_ID	符合JEDEC JEP-106标准的芯片标识码。
0x04	PACKAGE_SEL	封装选择指示符，0 = QFN80, 1 = QFN60
0x08	PLATFORM	平台寄存器。允许软件在预生产阶段识别其运行环境。生产后，PLATFORM始终为ASIC，非SIM。
0x14	GITREF_RP2350	芯片源码的Git哈希值。用于识别芯片版本。

### SYSINFO：CHIP\_ID 寄存器

偏移: 0x00

#### 说明

符合JEDEC JEP-106标准的芯片标识码。

表 1307. CHIP\_ID 寄存器

位	描述	类型	复位
31:28	修订版	只读	-
27:12	零件	只读	-
11:1	制造商	只读	-
0	STOP_BIT	只读	0x1

### SYSINFO：PACKAGE\_SEL 寄存器

偏移: 0x04

表 1308。PACKAGE\_SEL 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	封装选择指示符，0 = QFN80, 1 = QFN60	只读	0x0

### SYSINFO：PLATFORM 寄存器

偏移: 0x08

#### 说明

平台寄存器。允许软件在生产前开发阶段识别其运行环境。

生产后，PLATFORM 始终为 ASIC，非模拟。

表 1309。PLATFORM 寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	门级仿真	只读	-
3	批次仿真	只读	-
2	HDL 仿真	只读	-
1	ASIC	只读	-

位	描述	类型	复位
0	FPGA	只读	-

## SYSINFO: GITREF\_RP2350 寄存器

偏移: 0x14

表 1310。  
GITREF\_RP2350  
寄存器

位	描述	类型	复位
31:0	芯片源码的Git哈希值。用于识别芯片版本。	只读	-

## 12.15.2. SYSCFG

### 12.15.2.1. 概述

系统配置块控制杂项芯片设置，包括：

- 检查两个核心的调试停止状态
- 处理器 GPIO 输入同步器控制（设置为 1以允许输入同步器绕过，从而减少同步时钟延迟）
- 芯片内部的 SWD 接口控制（允许一个核心调试另一个核心，可能使调试连接更便捷）
- 状态保持内存断电（SRAM 外围设备在不使用时可断电以节省少量功耗）
  - 以此方式断电时，仍会向 SRAM 存储阵列供电；使用电源管理器（第6章）可完全切断电源
- AUXCTRL 寄存器中的其他控制

### 12.15.2.2. 相较于 RP2040 的变更

- 已将 NMI 掩码移动至 EPPB 中的每核寄存器（见第3.7.5.1节）。新寄存器在处理器热重置时复位，从而避免了启动只读存储器早期启动过程中 NMI 触发的问题。
- 扩展MEMPOWERDOWN以涵盖新的内存块
- 从DBGFORCE中移除控制项以适应新的单DP调试拓扑结构

### 12.15.2.3. 寄存器列表

系统配置寄存器起始基址为 **0x40008000**（在SDK中定义为SYSCFG\_BASE）。

表1311。SYSCFG  
寄存器列表

偏移量	名称	说明
0x00	PROC_CONFIG	处理器配置

偏移量	名称	说明
0x04	PROC_IN_SYNC_BYPASS	对于每一位，若为1，则绕过该GPIO与SIO中GPIO输入寄存器之间的输入同步器。 输入同步器通常不应被绕过，以避免向处理器引入亚稳态。 一般建议不绕过输入同步器，以防止向处理器注入亚稳态。 如有胆识，可绕过该同步器以节省两周期的输入延迟。 延迟。该寄存器适用于GPIO 0至31。
0x08	PROC_IN_SYNC_BYPASS_HI	对于每一位，若为1，则绕过该GPIO与SIO中GPIO输入寄存器之间的输入同步器。 输入同步器通常不应被绕过，以避免向处理器引入亚稳态。 一般建议不绕过输入同步器，以防止向处理器注入亚稳态。 如有胆识，可绕过该同步器以节省两周期的输入延迟。 延迟。该寄存器适用于GPIO 32至47及USB GPIO 56至57。 QSPI GPIO 58至63
0x0c	DBGFORCE	直接控制芯片的SWD调试端口
0x10	MEMPOWERDOWN	控制存储器的电源降压（PD）引脚。 置高电平可使存储器进入低功耗状态。在此状态下，存储器内容将被保留，但无法访问。 请谨慎使用。
0x14	AUXCTRL	辅助系统控制寄存器

## SYSCFG: PROC\_CONFIG 寄存器

偏移: 0x00

### 说明

处理器配置

表 1312。  
PROC\_CONFIG  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	PROC1_HALTED：指示 proc1 已停止运行	只读	0x0
0	PROC0_HALTED：指示 proc0 已停止运行	只读	0x0

## SYSCFG: PROC\_IN\_SYNC\_BYPASS 寄存器

偏移: 0x04

### 描述

对于每个位，为1时表示绕过该GPIO与SIO中GPIO输入寄存器之间的输入同步器。一般不应绕过输入同步器，以避免将亚稳态注入处理器。

如果您胆大，可以绕过以节省两个周期的输入延迟。该寄存器适用于GPIO 0至31。

表 1313。  
PROC\_IN\_SYNC\_BYPA  
SS 寄存器

位	描述	类型	复位
31:0	<b>GPIO</b>	读写	0x00000000

## SYSCFG：PROC\_IN\_SYNC\_BYPASS\_HI 寄存器

偏移: 0x08

### 说明

对于每个位，为1时表示绕过该 GPIO 与 SIO 中 GPIO 输入寄存器之间的输入同步器。一般不应绕过输入同步器，以避免将亚稳态注入处理器。

如果您胆大，可以绕过以节省两个周期的输入延迟。该寄存器适用于 GPIO 32...47。USB GPIO 56..57 QSPI GPIO 58..63

表 1314。  
PROC\_IN\_SYNC\_BYPA  
SS\_HI 寄存器

位	描述	类型	复位
31:28	<b>QSPI_SD</b>	读写	0x0
27	<b>QSPI_CSN</b>	读写	0x0
26	<b>QSPI_SCK</b>	读写	0x0
25	<b>USB_DM</b>	读写	0x0
24	<b>USB_DP</b>	读写	0x0
23:16	保留。	-	-
15:0	<b>GPIO</b>	读写	0x0000

## SYSCFG：DBGFORCE 寄存器

偏移: 0x0c

### 描述

直接控制芯片的SWD调试端口

表 1315。  
DBGFORCE 寄存器

位	描述	类型	复位
31:4	保留。	-	-
3	<b>ATTACH</b> : 将芯片调试端口连接至 syscfg 控制，同时断开其与外部 SWD 焊盘的连接。	读写	0x0
2	<b>SWCLK</b> : 当 ATTACH 设置时，直接驱动 SWCLK。	读写	0x1
1	<b>SWDI</b> : 当 ATTACH 设置时，直接驱动 SWDIO 输入。	读写	0x1
0	<b>SWDO</b> : 监视 SWDIO 输出的数值。	只读	-

## SYSCFG：MEMPOWERDOWN 寄存器

偏移: 0x10

### 描述

控制存储器的电源降压（PD）引脚。

置高电平可使存储器进入低功耗状态。在该状态下，内存内容将被保留但不可访问，使用时须谨慎。

表 1316。  
MEMPOWERDOWN  
寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<b>BOOTRAM</b>	读写	0x0

位	描述	类型	复位
11	<b>只读存储器</b>	读写	0x0
10	<b>USB</b>	读写	0x0
9	<b>SRAM9</b>	读写	0x0
8	<b>SRAM8</b>	读写	0x0
7	<b>SRAM7</b>	读写	0x0
6	<b>SRAM6</b>	读写	0x0
5	<b>SRAM5</b>	读写	0x0
4	<b>SRAM4</b>	读写	0x0
3	<b>SRAM3</b>	读写	0x0
2	<b>SRAM2</b>	读写	0x0
1	<b>SRAM1</b>	读写	0x0
0	<b>SRAM0</b>	读写	0x0

## SYSCFG：AUXCTRL 寄存器

偏移：0x14

### 说明

辅助系统控制寄存器

表 1317。AUXCTRL 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	* 位 7:3：保留 * 位 2：设置以屏蔽 OTP 电源模拟电源检测，防止 OTP 控制器与 PSM 被重置。 * 位 1：清除时，LPOSC 输出将被异或加入 TRNG ROSC 输出，作为额外的非相关熵源。设置后，该行为将被禁用。 * 位 0：通过断言其 WDRESET 输入，强制 POWMAN 时钟切换至 LPOSC。在启动包含 CLOCKS 阶段的 RSM 看门狗复位之前，且当看门狗复位发生时 POWMAN 正从 clk_ref 运行，必须设置此项。否则，CLOCKS 模块复位时在 clk_ref 上产生的短脉冲可能会影响 POWMAN 寄存器状态。	读写	0x00

### 12.15.3. TBMAN

TBMAN 指测试平台管理器，在芯片开发仿真过程中用于验证设计。在这些仿真中，TBMAN 允许运行于 RP2350 上的软件控制测试平台及仿真环境。在实际芯片中，其唯一作用是提供一个 PLATFORM 寄存器，用以标示这是实际芯片。该 PLATFORM 功能在 sysinfo（第12.15.1节）寄存器中亦有重复实现。

### 12.15.3.1. 寄存器列表

TBMAN 寄存器起始地址为 `0x40160000` (SDK 中定义为 TBMAN\_BASE)。

表1318.  
TBMAN寄存器列表

偏移量	名称	说明
0x0	PLATFORM	指示所使用的平台类型

## TBMAN：PLATFORM寄存器

偏移：0x0

### 描述

指示所使用的平台类型

表1319.  
PLATFORM寄存器

位	描述	类型	复位
31:3	保留。	-	-
2	<b>HDLSIM</b> ：表示该平台为仿真平台	只读	0x0
1	<b>FPGA</b> ：表示该平台为现场可编程门阵列	只读	0x0
0	<b>ASIC</b> ：表示该平台为专用集成电路	只读	0x1

## 12.15.4. BUSCTRL

本模块提供系统总线结构的基础控制与监控功能。

### 12.15.4.1. 总线优先级

RP2350实现了第2.1.1节所述的动态总线优先级方案。BUS\_PRIORITY寄存器执行该方案的优先级控制。

### 12.15.4.2. 性能计数器

共有四个24位计数器，每个计数器可订阅系统总线结构中的单一性能事件。

计数器在全1值时达到饱和：计数器达到最大值后停止递增，且不回绕至零。

性能计数器初始处于禁用状态：必须向PERFCTR\_EN写入 `1`，计数器方可开始递增。在运行受分析代码段之前，写入任意值以将计数器清零，并在进入该代码段前立即启用计数器。在离开受分析代码段时，立即再次禁用计数器。计数器不支持任意写入：仅能从零开始递增。

向性能事件选择寄存器PERFSEL0至PERFSEL3写入数据，以选择使对应计数器PERFCTR0至PERFCTR3递增的性能事件。

对于主系统AHB5交叉开关图5所示的十七个下游总线端口，每个端口的性能计数器可检测四种事件类型。这些事件不区分读写操作，但能区分不同类型的总线阻塞，有助于性能问题的诊断。事件类型包括：

#### 访问权限

任何访问在该下游端口完成时递增计数。

#### 争用访问

任何先前因端口被

另一访问占用而阻塞的访问在该下游端口完成时递增计数。例如，如果两个管理器同时访问一个初始空闲的端口，其中一个访问将先完成。先完成的访问称为无竞争访问，不会增加此计数器。第二个完成的访问（因另一个管理器的访问最初被延迟）被视为有竞争访问，并在完成时增加此计数器。

#### 上游阻塞周期

只要任何管理器在该端口发生阻塞，每个周期递增一次。这可能是由于与其他管理器的仲裁（即有竞争访问）或下游总线端口的阻塞，例如访问慢速外设。此统计测量位置为端口处，在离开主AHB5交叉开关之前。

#### 下游阻塞周期

只要该端口自身在下游总线上发生阻塞，每个周期递增一次。这表明外设或存储设备响应缓慢，如XIP缓存未命中。

上述前两种事件类型与RP2040相同，后两种则为RP2350新增。

#### 12.15.4.3. 寄存器列表

总线结构寄存器起始基址为 [0x40068000](#)（在SDK中定义为BUSCTRL\_BASE）。

表1320。BUSCTRL寄存器列表

偏移量	名称	说明
0x00	<a href="#">BUS_PRIORITY</a>	设置各主设备的总线仲裁优先级。
0x04	<a href="#">BUS_PRIORITY_ACK</a>	总线优先级确认
0x08	<a href="#">PERFCTR_EN</a>	启用性能计数器。若为0，性能计数器将不递增。该功能可用于精确启动或停止事件采样，以涵盖被分析代码的相关段落。  性能计数器初始为禁用状态，以节约能源。
0x0c	<a href="#">PERFCTR0</a>	总线结构性能计数器0
0x10	<a href="#">PERFSEL0</a>	PERFCTR0的总线结构性能事件选择器
0x14	<a href="#">PERFCTR1</a>	总线结构性能计数器1
0x18	<a href="#">PERFSEL1</a>	PERFCTR1的总线结构性能事件选择器
0x1c	<a href="#">PERFCTR2</a>	总线结构性能计数器2
0x20	<a href="#">PERFSEL2</a>	PERFCTR2的总线结构性能事件选择
0x24	<a href="#">PERFCTR3</a>	总线结构性能计数器3
0x28	<a href="#">PERFSEL3</a>	PERFCTR3的总线结构性能事件选择

#### BUSCTRL: BUS\_PRIORITY 寄存器

偏移: 0x00

##### 说明

设置各主设备的总线仲裁优先级。

表 1321  
BUS\_PRIORITY 寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<b>DMA_W</b> : 0 - 低优先级, 1 - 高优先级	读写	0x0
11:9	保留。	-	-
8	<b>DMA_R</b> : 0 - 低优先级, 1 - 高优先级	读写	0x0

位	描述	类型	复位
7:5	保留。	-	-
4	<b>PROC1</b> : 0 - 低优先级, 1 - 高优先级	读写	0x0
3:1	保留。	-	-
0	<b>PROC0</b> : 0 - 低优先级, 1 - 高优先级	读写	0x0

## BUSCTRL: BUS\_PRIORITY\_ACK 寄存器

偏移: 0x04

### 说明

总线优先级确认

表 1322。  
BUS\_PRIORITY\_ACK 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	所有仲裁者登记新的全局优先级后，该值变为1。 仲裁者在处理新的非顺序访问时会更新其本地优先级。 在正常情况下，该过程几乎立即发生。	只读	0x0

## BUSCTRL: PERFCTR\_EN 寄存器

偏移: 0x08

表 1323。  
PERFCTR\_EN 寄存器

位	描述	类型	复位
31:1	保留。	-	-
0	启用性能计数器。若为0，性能计数器不递增。可用于精确控制代码采样分析区段的事件采样启动与停止。  性能计数器初始为禁用状态，以节约能源。	读写	0x0

## BUSCTRL: PERFCTR0 寄存器

偏移: 0x0c

### 描述

总线结构性能计数器0

表 1324。  
PERFCTR0 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	总线结构性能计数器0 当PERFCTR_EN置位时，计数总线结构仲裁者的特定事件信号。 写入任意值以清零。使用PERFSEL0选择要计数的事件。	WC	0x000000

## BUSCTRL: PERFSEL0 寄存器

偏移: 0x10

### 描述

PERFCTR0的总线结构性能事件选择器

表1325。PERFSEL0  
寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:0	为PERFCTR0选择一个事件。对于主交叉开关的每个下游端口，有四种事件可选：ACCESS，表示发生了一次访问；ACCESS_CONTESTED，表示发生了一次由于其他主控器争用而导致先前暂停的访问；STALL_DOWNSTREAM，统计任何主控器因下游总线暂停而停滞的周期数；STALL_UPSTREAM，统计任何主控器因任何原因（包括其他主控器争用）停滞的周期数。	读写	0x1f
	枚举值：		
	0x00 → SIOB PROC1_STALL_UPSTREAM		
	0x01 → SIOB PROC1_STALL_DOWNSTREAM		
	0x02 → SIOB PROC1_ACCESS_CONTESTED		
	0x03 → SIOB PROC1_ACCESS		
	0x04 → SIOB PROC0_STALL_UPSTREAM		
	0x05 → SIOB PROC0_STALL_DOWNSTREAM		
	0x06 → SIOB PROC0_ACCESS_CONTESTED		
	0x07 → SIOB PROC0_ACCESS		
	0x08 → APB_STALL_UPSTREAM		
	0x09 → APB_STALL_DOWNSTREAM		
	0x0a → APB_ACCESS_CONTESTED		
	0x0b → APB_ACCESS		
	0x0c → FASTPERI_STALL_UPSTREAM		
	0x0d → FASTPERI_STALL_DOWNSTREAM		
	0x0e → FASTPERI_ACCESS_CONTESTED		
	0x0f → FASTPERI_ACCESS		
	0x10 → SRAM9_STALL_UPSTREAM		
	0x11 → SRAM9_STALL_DOWNSTREAM		
	0x12 → SRAM9_ACCESS_CONTESTED		
	0x13 → SRAM9_ACCESS		
	0x14 → SRAM8_STALL_UPSTREAM		
	0x15 → SRAM8_STALL_DOWNSTREAM		
	0x16 → SRAM8_ACCESS_CONTESTED		
	0x17 → SRAM8_ACCESS		
	0x18 → SRAM7_STALL_UPSTREAM		
	0x19 → SRAM7_STALL_DOWNSTREAM		
	0x1a → SRAM7_ACCESS_CONTESTED		
	0x1b → SRAM7_ACCESS		
	0x1c → SRAM6_STALL_UPSTREAM		

位	描述	类型	复位
	0x1d → SRAM6_STALL_DOWNSTREAM		
	0x1e → SRAM6_ACCESS_CONTESTED		
	0x1f → SRAM6_ACCESS		
	0x20 → SRAM5_STALL_UPSTREAM		
	0x21 → SRAM5_STALL_DOWNSTREAM		
	0x22 → SRAM5_ACCESS_CONTESTED		
	0x23 → SRAM5_ACCESS		
	0x24 → SRAM4_STALL_UPSTREAM		
	0x25 → SRAM4_STALL_DOWNSTREAM		
	0x26 → SRAM4_ACCESS_CONTESTED		
	0x27 → SRAM4_ACCESS		
	0x28 → SRAM3_STALL_UPSTREAM		
	0x29 → SRAM3_STALL_DOWNSTREAM		
	0x2a → SRAM3_ACCESS_CONTESTED		
	0x2b → SRAM3_ACCESS		
	0x2c → SRAM2_STALL_UPSTREAM		
	0x2d → SRAM2_STALL_DOWNSTREAM		
	0x2e → SRAM2_ACCESS_CONTESTED		
	0x2f → SRAM2_ACCESS		
	0x30 → SRAM1_STALL_UPSTREAM		
	0x31 → SRAM1_STALL_DOWNSTREAM		
	0x32 → SRAM1_ACCESS_CONTESTED		
	0x33 → SRAM1_ACCESS		
	0x34 → SRAM0_STALL_UPSTREAM		
	0x35 → SRAM0_STALL_DOWNSTREAM		
	0x36 → SRAM0_ACCESS_CONTESTED		
	0x37 → SRAM0_ACCESS		
	0x38 → XIP_MAIN1_STALL_UPSTREAM		
	0x39 → XIP_MAIN1_STALL_DOWNSTREAM		
	0x3a → XIP_MAIN1_ACCESS_CONTESTED		
	0x3b → XIP_MAIN1_ACCESS		
	0x3c → XIP_MAIN0_STALL_UPSTREAM		
	0x3d → XIP_MAIN0_STALL_DOWNSTREAM		
	0x3e → XIP_MAIN0_ACCESS_CONTESTED		
	0x3f → XIP_MAIN0_ACCESS		
	0x40 → 只读存储器_STALL_UPSTREAM		

位	描述	类型	复位
	0x41 → 只读存储器_STALL_DOWNSTREAM		
	0x42 → 只读存储器_ACCESS_CONTESTED		
	0x43 → 只读存储器_ACCESS		

## BUSCTRL：PERFCTR1寄存器

偏移：0x14

### 说明

总线结构性能计数器1

表1326。  
PERFCTR1寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	总线结构饱和性能计数器1 当PERFCTR_EN置位时，计数总线结构仲裁者的特定事件信号。 写入任意值以清零。使用PERFSEL1选择计数事件	WC	0x000000

## BUSCTRL：PERFSEL1寄存器

偏移量：0x18

### 说明

PERFCTR1的总线结构性能事件选择器

表1327。PERFSEL1  
寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:0	为PERFCTR1选择事件。针对主交叉开关的每个下游端口，提供四种事件类型：ACCESS，表示发生了一次访问； ACCESS_CONTESTED，表示发生了一次由于其他主控器争用而导致先前暂停的访问；STALL_DOWNSTREAM，统计任何主控器因下游总线暂停而停滞的周期数；STALL_UPSTREAM，统计任何主控器因任何原因（包括其他主控器争用）停滞的周期数。	读写	0x1f
	枚举值：		
	0x00 → SIOB_PROC1_STALL_UPSTREAM		
	0x01 → SIOB_PROC1_STALL_DOWNSTREAM		
	0x02 → SIOB_PROC1_ACCESS_CONTESTED		
	0x03 → SIOB_PROC1_ACCESS		
	0x04 → SIOB_PROC0_STALL_UPSTREAM		
	0x05 → SIOB_PROC0_STALL_DOWNSTREAM		
	0x06 → SIOB_PROC0_ACCESS_CONTESTED		
	0x07 → SIOB_PROC0_ACCESS		
	0x08 → APB_STALL_UPSTREAM		
	0x09 → APB_STALL_DOWNSTREAM		
	0x0a → APB_ACCESS_CONTESTED		

位	描述	类型	复位
0x0b	→ APB_ACCESS		
0x0c	→ FASTPERI_STALL_UPSTREAM		
0x0d	→ FASTPERI_STALL_DOWNSTREAM		
0x0e	→ FASTPERI_ACCESS_CONTESTED		
0x0f	→ FASTPERI_ACCESS		
0x10	→ SRAM9_STALL_UPSTREAM		
0x11	→ SRAM9_STALL_DOWNSTREAM		
0x12	→ SRAM9_ACCESS_CONTESTED		
0x13	→ SRAM9_ACCESS		
0x14	→ SRAM8_STALL_UPSTREAM		
0x15	→ SRAM8_STALL_DOWNSTREAM		
0x16	→ SRAM8_ACCESS_CONTESTED		
0x17	→ SRAM8_ACCESS		
0x18	→ SRAM7_STALL_UPSTREAM		
0x19	→ SRAM7_STALL_DOWNSTREAM		
0x1a	→ SRAM7_ACCESS_CONTESTED		
0x1b	→ SRAM7_ACCESS		
0x1c	→ SRAM6_STALL_UPSTREAM		
0x1d	→ SRAM6_STALL_DOWNSTREAM		
0x1e	→ SRAM6_ACCESS_CONTESTED		
0x1f	→ SRAM6_ACCESS		
0x20	→ SRAM5_STALL_UPSTREAM		
0x21	→ SRAM5_STALL_DOWNSTREAM		
0x22	→ SRAM5_ACCESS_CONTESTED		
0x23	→ SRAM5_ACCESS		
0x24	→ SRAM4_STALL_UPSTREAM		
0x25	→ SRAM4_STALL_DOWNSTREAM		
0x26	→ SRAM4_ACCESS_CONTESTED		
0x27	→ SRAM4_ACCESS		
0x28	→ SRAM3_STALL_UPSTREAM		
0x29	→ SRAM3_STALL_DOWNSTREAM		
0x2a	→ SRAM3_ACCESS_CONTESTED		
0x2b	→ SRAM3_ACCESS		
0x2c	→ SRAM2_STALL_UPSTREAM		
0x2d	→ SRAM2_STALL_DOWNSTREAM		
0x2e	→ SRAM2_ACCESS_CONTESTED		

位	描述	类型	复位
	0x2f → SRAM2_ACCESS		
	0x30 → SRAM1_STALL_UPSTREAM		
	0x31 → SRAM1_STALL_DOWNSTREAM		
	0x32 → SRAM1_ACCESS_CONTESTED		
	0x33 → SRAM1_ACCESS		
	0x34 → SRAM0_STALL_UPSTREAM		
	0x35 → SRAM0_STALL_DOWNSTREAM		
	0x36 → SRAM0_ACCESS_CONTESTED		
	0x37 → SRAM0_ACCESS		
	0x38 → XIP_MAIN1_STALL_UPSTREAM		
	0x39 → XIP_MAIN1_STALL_DOWNSTREAM		
	0x3a → XIP_MAIN1_ACCESS_CONTESTED		
	0x3b → XIP_MAIN1_ACCESS		
	0x3c → XIP_MAIN0_STALL_UPSTREAM		
	0x3d → XIP_MAIN0_STALL_DOWNSTREAM		
	0x3e → XIP_MAIN0_ACCESS_CONTESTED		
	0x3f → XIP_MAIN0_ACCESS		
	0x40 → 只读存储器_STALL_UPSTREAM		
	0x41 → 只读存储器_STALL_DOWNSTREAM		
	0x42 → 只读存储器_ACCESS_CONTESTED		
	0x43 → 只读存储器_ACCESS		

## BUSCTRL：PERFCTR2寄存器

偏移：0x1c

### 说明

总线结构性能计数器 2

表1328。  
PERFCTR2寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	总线结构饱和性能计数器2 当PERFCTR_EN置位时，计数总线结构仲裁者的特定事件信号。 写入任意值以清零。使用PERFSEL2选择要计数的事件	WC	0x000000

## BUSCTRL：PERFSEL2寄存器

偏移：0x20

### 说明

PERFCTR2 的总线结构性能事件选择

表1329. *PERFSEL2*  
寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:0	为PERFCTR2选择事件。针对主交叉开关的每个下游端口，提供四种事件类型：ACCESS，表示发生了一次访问； ACCESS_CONTESTED，表示发生了一次由于其他主控器争用而导致先前暂停的访问；STALL_DOWNSTREAM，统计任何主控器因下游总线暂停而停滞的周期数；STALL_UPSTREAM，统计任何主控器因任何原因（包括其他主控器争用）停滞的周期数。	读写	0x1f
	枚举值：		
	0x00 → SIOB PROC1_STALL_UPSTREAM		
	0x01 → SIOB PROC1_STALL_DOWNSTREAM		
	0x02 → SIOB PROC1_ACCESS_CONTESTED		
	0x03 → SIOB PROC1_ACCESS		
	0x04 → SIOB PROC0_STALL_UPSTREAM		
	0x05 → SIOB PROC0_STALL_DOWNSTREAM		
	0x06 → SIOB PROC0_ACCESS_CONTESTED		
	0x07 → SIOB PROC0_ACCESS		
	0x08 → APB_STALL_UPSTREAM		
	0x09 → APB_STALL_DOWNSTREAM		
	0x0a → APB_ACCESS_CONTESTED		
	0x0b → APB_ACCESS		
	0x0c → FASTPERI_STALL_UPSTREAM		
	0x0d → FASTPERI_STALL_DOWNSTREAM		
	0x0e → FASTPERI_ACCESS_CONTESTED		
	0x0f → FASTPERI_ACCESS		
	0x10 → SRAM9_STALL_UPSTREAM		
	0x11 → SRAM9_STALL_DOWNSTREAM		
	0x12 → SRAM9_ACCESS_CONTESTED		
	0x13 → SRAM9_ACCESS		
	0x14 → SRAM8_STALL_UPSTREAM		
	0x15 → SRAM8_STALL_DOWNSTREAM		
	0x16 → SRAM8_ACCESS_CONTESTED		
	0x17 → SRAM8_ACCESS		
	0x18 → SRAM7_STALL_UPSTREAM		
	0x19 → SRAM7_STALL_DOWNSTREAM		
	0x1a → SRAM7_ACCESS_CONTESTED		
	0x1b → SRAM7_ACCESS		
	0x1c → SRAM6_STALL_UPSTREAM		

位	描述	类型	复位
	0x1d → SRAM6_STALL_DOWNSTREAM		
	0x1e → SRAM6_ACCESS_CONTESTED		
	0x1f → SRAM6_ACCESS		
	0x20 → SRAM5_STALL_UPSTREAM		
	0x21 → SRAM5_STALL_DOWNSTREAM		
	0x22 → SRAM5_ACCESS_CONTESTED		
	0x23 → SRAM5_ACCESS		
	0x24 → SRAM4_STALL_UPSTREAM		
	0x25 → SRAM4_STALL_DOWNSTREAM		
	0x26 → SRAM4_ACCESS_CONTESTED		
	0x27 → SRAM4_ACCESS		
	0x28 → SRAM3_STALL_UPSTREAM		
	0x29 → SRAM3_STALL_DOWNSTREAM		
	0x2a → SRAM3_ACCESS_CONTESTED		
	0x2b → SRAM3_ACCESS		
	0x2c → SRAM2_STALL_UPSTREAM		
	0x2d → SRAM2_STALL_DOWNSTREAM		
	0x2e → SRAM2_ACCESS_CONTESTED		
	0x2f → SRAM2_ACCESS		
	0x30 → SRAM1_STALL_UPSTREAM		
	0x31 → SRAM1_STALL_DOWNSTREAM		
	0x32 → SRAM1_ACCESS_CONTESTED		
	0x33 → SRAM1_ACCESS		
	0x34 → SRAM0_STALL_UPSTREAM		
	0x35 → SRAM0_STALL_DOWNSTREAM		
	0x36 → SRAM0_ACCESS_CONTESTED		
	0x37 → SRAM0_ACCESS		
	0x38 → XIP_MAIN1_STALL_UPSTREAM		
	0x39 → XIP_MAIN1_STALL_DOWNSTREAM		
	0x3a → XIP_MAIN1_ACCESS_CONTESTED		
	0x3b → XIP_MAIN1_ACCESS		
	0x3c → XIP_MAIN0_STALL_UPSTREAM		
	0x3d → XIP_MAIN0_STALL_DOWNSTREAM		
	0x3e → XIP_MAIN0_ACCESS_CONTESTED		
	0x3f → XIP_MAIN0_ACCESS		
	0x40 → 只读存储器_STALL_UPSTREAM		

位	描述	类型	复位
	0x41 → 只读存储器_STALL_DOWNSTREAM		
	0x42 → 只读存储器_ACCESS_CONTESTED		
	0x43 → 只读存储器_ACCESS		

## BUSCTRL：PERFCTR3寄存器

偏移量：0x24

### 说明

总线结构性能计数器 3

表1330。  
PERFCTR3寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	总线结构饱和性能计数器3 当PERFCTR_EN置位时，计数总线结构仲裁者的特定事件信号。 写入任意值以清零。使用PERFSEL3选择要计数的事件	WC	0x000000

## BUSCTRL：PERFSEL3寄存器

偏移：0x28

### 说明

PERFCTR3 的总线结构性能事件选择

表1331. PERFSEL3  
寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:0	为PERFCTR3选择事件。针对主交叉开关的每个下游端口，提供四种事件类型：ACCESS，表示发生了一次访问； ACCESS_CONTESTED，表示发生了一次由于其他主控器争用而导致先前暂停的访问；STALL_DOWNSTREAM，统计任何主控器因下游总线暂停而停滞的周期数；STALL_UPSTREAM，统计任何主控器因任何原因（包括其他主控器争用）停滞的周期数。	读写	0x1f
	枚举值：		
	0x00 → SIOB PROC1_STALL_UPSTREAM		
	0x01 → SIOB PROC1_STALL_DOWNSTREAM		
	0x02 → SIOB PROC1_ACCESS_CONTESTED		
	0x03 → SIOB PROC1_ACCESS		
	0x04 → SIOB PROC0_STALL_UPSTREAM		
	0x05 → SIOB PROC0_STALL_DOWNSTREAM		
	0x06 → SIOB PROC0_ACCESS_CONTESTED		
	0x07 → SIOB PROC0_ACCESS		
	0x08 → APB_STALL_UPSTREAM		
	0x09 → APB_STALL_DOWNSTREAM		
	0x0a → APB_ACCESS_CONTESTED		

位	描述	类型	复位
0x0b	→ APB_ACCESS		
0x0c	→ FASTPERI_STALL_UPSTREAM		
0x0d	→ FASTPERI_STALL_DOWNSTREAM		
0x0e	→ FASTPERI_ACCESS_CONTESTED		
0x0f	→ FASTPERI_ACCESS		
0x10	→ SRAM9_STALL_UPSTREAM		
0x11	→ SRAM9_STALL_DOWNSTREAM		
0x12	→ SRAM9_ACCESS_CONTESTED		
0x13	→ SRAM9_ACCESS		
0x14	→ SRAM8_STALL_UPSTREAM		
0x15	→ SRAM8_STALL_DOWNSTREAM		
0x16	→ SRAM8_ACCESS_CONTESTED		
0x17	→ SRAM8_ACCESS		
0x18	→ SRAM7_STALL_UPSTREAM		
0x19	→ SRAM7_STALL_DOWNSTREAM		
0x1a	→ SRAM7_ACCESS_CONTESTED		
0x1b	→ SRAM7_ACCESS		
0x1c	→ SRAM6_STALL_UPSTREAM		
0x1d	→ SRAM6_STALL_DOWNSTREAM		
0x1e	→ SRAM6_ACCESS_CONTESTED		
0x1f	→ SRAM6_ACCESS		
0x20	→ SRAM5_STALL_UPSTREAM		
0x21	→ SRAM5_STALL_DOWNSTREAM		
0x22	→ SRAM5_ACCESS_CONTESTED		
0x23	→ SRAM5_ACCESS		
0x24	→ SRAM4_STALL_UPSTREAM		
0x25	→ SRAM4_STALL_DOWNSTREAM		
0x26	→ SRAM4_ACCESS_CONTESTED		
0x27	→ SRAM4_ACCESS		
0x28	→ SRAM3_STALL_UPSTREAM		
0x29	→ SRAM3_STALL_DOWNSTREAM		
0x2a	→ SRAM3_ACCESS_CONTESTED		
0x2b	→ SRAM3_ACCESS		
0x2c	→ SRAM2_STALL_UPSTREAM		
0x2d	→ SRAM2_STALL_DOWNSTREAM		
0x2e	→ SRAM2_ACCESS_CONTESTED		

位	描述	类型	复位
	0x2f → SRAM2_ACCESS		
	0x30 → SRAM1_STALL_UPSTREAM		
	0x31 → SRAM1_STALL_DOWNSTREAM		
	0x32 → SRAM1_ACCESS_CONTESTED		
	0x33 → SRAM1_ACCESS		
	0x34 → SRAM0_STALL_UPSTREAM		
	0x35 → SRAM0_STALL_DOWNSTREAM		
	0x36 → SRAM0_ACCESS_CONTESTED		
	0x37 → SRAM0_ACCESS		
	0x38 → XIP_MAIN1_STALL_UPSTREAM		
	0x39 → XIP_MAIN1_STALL_DOWNSTREAM		
	0x3a → XIP_MAIN1_ACCESS_CONTESTED		
	0x3b → XIP_MAIN1_ACCESS		
	0x3c → XIP_MAIN0_STALL_UPSTREAM		
	0x3d → XIP_MAIN0_STALL_DOWNSTREAM		
	0x3e → XIP_MAIN0_ACCESS_CONTESTED		
	0x3f → XIP_MAIN0_ACCESS		
	0x40 → 只读存储器_STALL_UPSTREAM		
	0x41 → 只读存储器_STALL_DOWNSTREAM		
	0x42 → 只读存储器_ACCESS_CONTESTED		
	0x43 → 只读存储器_ACCESS		

# 第13章 一次性可编程存储器 (OTP)

RP2350 提供8 kB的一次性可编程存储 (OTP) , 其内容包括:

- 预编程的每个设备信息, 例如唯一设备标识符和振荡器校准值
- 安全配置, 如禁用调试功能和启用安全启动
- 用于安全启动的公钥指纹
- 用于将闪存内容解密到SRAM的对称密钥
- USB引导加载程序的配置, 例如自定义VID/PID及描述符
- 可启动的软件镜像, 适用于低成本无闪存应用或自定义引导加载程序
- 任何其他用户定义的数据, 如每个设备的个性化参数

有关预定义OTP内容的完整列表, 请参见第13.10节。

OTP物理上由4096行24位组成的阵列构成。您可以直接访问这些24位数值, 但硬件同时支持在每行存储16位数据, 配备6位海明码ECC保护和2位比特极性反转保护, 从而实现8192字节的ECC数据容量。

在一台空白设备上, OTP内容全部为零, 除了一些在制造测试期间预先编程的基本设备信息。每个位均可不可逆地由零编程为一。编程OTP内容的步骤如下:

- 通过SBPI桥接器直接访问寄存器
- 调用bootrom的 `otp_access` API (第5.4.8.21节)
- 使用USB引导加载程序的PICOBLOCK接口 (第5.6节)

RP2350针对OTP实施基于页面的权限控制, 以划分安全区与非安全区数据, 并确保不应更改的内容不会被更改。OTP地址空间逻辑上划分为64页, 每页64行大小, 每页包含128字节的ECC数据。各页面初始具备完全读写权限, 但可针对安全区、非安全区及引导加载程序访问分别限制为只读或不可访问。

页面权限信息本身存储于OTP中。以此方式锁定页面是一种不可逆操作, 称为硬锁定。硬件还支持软锁定, 即通过向SW\_LOCK0至SW\_LOCK63中的相应寄存器写入值, 进一步限制页面权限; 此限制将持续有效, 直至下一次OTP重置。

重置OTP块同时也会重置处理器, 因此软锁定可用于限制敏感内容 (例如解密密钥) 仅在早期引导阶段可用。

OTP访问密钥 (第13.5.2节) 提供了额外的安全保护层。一个固定挑战值被写入只写OTP区域。注册到该密钥的页面必须将密钥输入到只写寄存器, 方可开启读写访问权限。该机制支持配置数据, 允许板载制造商访问或编辑, 但禁止设备上运行的一般固件访问。

## 13.1. OTP 地址映射

OTP硬件位于起始地址 `0x40120000` (SDK中的OTP\_BASE) 起始的128 kB区域。地址第 16位用于选择该区域内的OTP控制寄存器 (低64 kB) 或OTP读取数据别名 (高64 kB) 。

OTP 控制寄存器 (第13.9节) 以4 kB间隔进行别名映射, 以实现第2.1.3节中描述的常规置位、清除及异或 (XOR) 原子写入别名。

从 `0x40130000` 开始的读取数据区域进一步划分为四个别名:

- `0x40130000`, `OTP_DATA_BASE`: ECC读取别名。进行32位读取时, 将返回两个相邻行经ECC校正的数据, 权限不符时返回全1。仅填充了首8 kB。

- **0x40138000, OTP\_DATA\_GUARDED\_BASE**: ECC受保护读取别名。成功读取时返回与 **OTP\_DATA\_BASE** 相同的数据。仅填充了首 8 kB。
- **0x40134000, OTP\_DATA\_RAW\_BASE**: 原始读取别名。进行 32 位读取时，直接返回单行的 24 位内容，最高 8 位为零；权限不符时返回全 1。
- **0x4013c000,OTP\_DATA\_RAW\_GUARDED\_BASE**: 原始的，受保护读取别名。成功读取返回与 **OTP\_DATA\_RAW\_BASE** 相同的数据。

地址的第 14 位选择 ECC (0) 或原始数据 (1)。地址的第 15 位选择非保护访问 (0) 或保护访问 (1)。受保护读取返回与非保护读取相同的数据，但会执行额外的硬件一致性检查，并在权限失败时返回总线故障。更多信息，请参见第 13.1.1 节。

#### **!** 重要

起始地址为 **0x40130000** 的读取数据区域仅在 USR.DCTRL 设置时可访问，否则所有读取操作均返回总线错误。当通过 SBPI 桥对 OTP 编程时，该位被清除。

写入读取数据别名不是有效操作，且始终返回总线故障。OTP 由 SBPI 桥编程，该桥由 bootrom 内部使用的 **otp\_access** 接口控制，详见第 5.4.8.21 节。

### 13.1.1. 保护读取

受保护别名的读取与非受保护读取在以下方面存在差异：

- 权限失败时返回总线故障，而非全为 1 的位模式。
- 检测到不可纠正的 ECC 错误时，将返回总线故障。
- 受保护读取会执行额外的硬件一致性检查以检测电源瞬态。若该检查失败，读取操作将返回总线故障。

这些检查有助于在存在故意故障注入可能性的环境中，使 OTP 实现容错安全。例如，RP2350 BootROM 使用受保护读取检查启动配置标志。

成功的受保护读取返回的数据，与相应非受保护别名的成功读取返回的数据相同。

#### **!** 重要

在安全环境中依赖 OTP 数据的用户应始终执行受保护读取，且强烈建议使用 ECC。对于无法使用 ECC 的行，软件应确保多次重叠读取的数据一致性。

## 13.2. 背景：OTP IP 细节

RP2350 OTP 子系统采用 Synopsys NVM OTP IP，该 IP 包括三部分：

- 集成电源供应 (IPS)，包括：
  - 电荷泵（用于编程）
  - 稳压器（用于读取）
- OTP 宏单元 (SHF, 保险丝)
  - 4096×24 (8 kB, 含 ECC, 16 位 ECC 写入粒度)
- 访问端口 (AP)，提供：

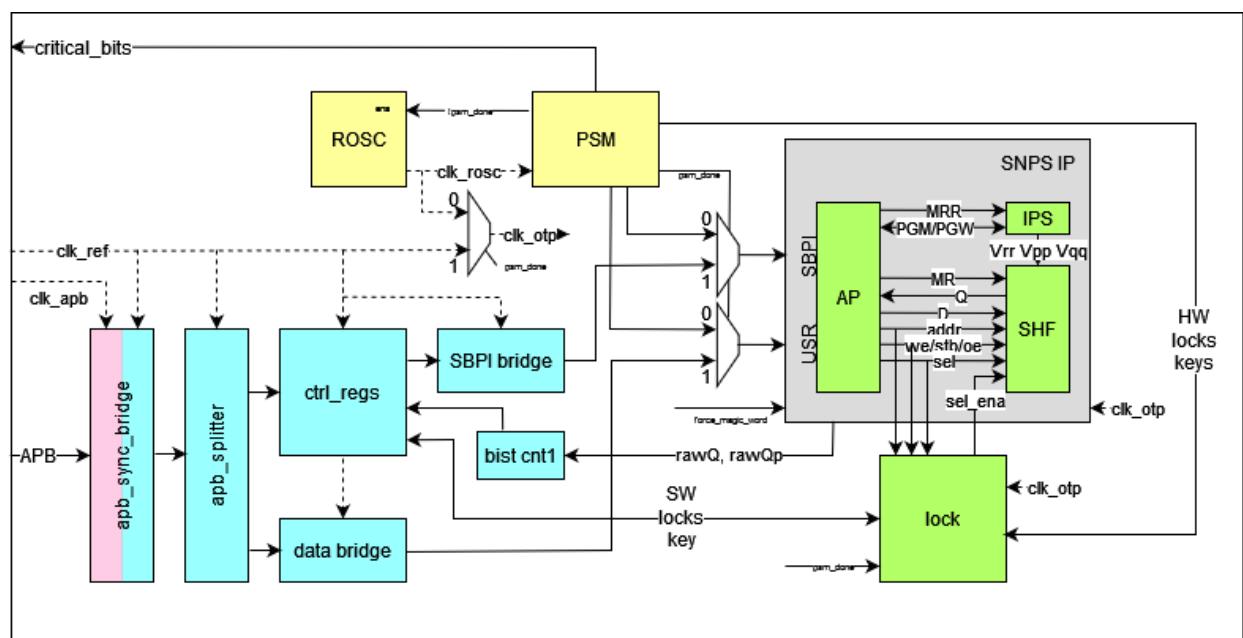
- 基本读取访问
- 编程访问
- ECC 及位冗余
- BOOT 功能，在系统启动时轮询稳定的 OTP 电源

### 13.3. 背景：OTP 硬件架构

本图展示了三个 Synopsys IP 组件的集成情况，以及为使其在 RP2350 系统和安全架构上下文中正常运行而添加的 Raspberry Pi 硬件。具体包括：

- 用于连接 SoC 的 APB 接口
- 内部环形振荡器，带时钟边沿随机化
- 启动状态机，基于环形振荡器运行
- 锁定衬套，位于 SNPS RTL 与存储核心（保险丝）之间

图 142. OTP 架构



OTP 子系统时钟在硬件启动期间最初由 OTP 启动振荡器（第 13.3.3 节）提供，但在处理器运行任何软件之前切换至 `clk_rf`。访问OTP时，`clk_ref`的频率不得超过25 MHz。

#### 13.3.1. 锁存衬垫

锁定衬垫插入于Synopsys AP模块与SHF模块之间，用于基于以下条件强制执行读写页锁定：

- AP → SHF 总线上呈现的OTP地址
- AP → SHF 总线上的读写选通信号
- 导致该SHF访问的上游总线访问的安全属性（如当前通过USR.DCTRL启用SBPI，则视为安全访问）

鉴于Synopsys AP在编程OTP行时既执行读操作又执行写操作，故无法禁用对某地址的读取而不同禁用写入。支持三种锁定状态：

- 读/写
- 只读
- 不可访问

完整的锁定方案详述于第13.5节，简述如下：

- 每个OTP页的锁定状态在启动时从OTP中读取。
- 针对安全/非安全访问分别保存独立的锁定状态副本。锁片对安全读取应用安全读取权限，对非安全读取应用非安全读取权限。写入操作无此规则，因非安全代码无法访问编程硬件。
- OTP存储中的锁定编码保证页面只能锁定至更严格状态（顺序为RW → RO → 不可访问），且绝不允许回退至较宽松状态。
- 软件可在运行时提升各单独锁状态，且无需重新编程OTP，该状态持续至OTP PSM重新执行。
- 软件锁同样遵循锁定进阶顺序（RW → RO → 不可访问），且不可回退。

完整锁定方案详见第13.5节。

### 13.3.2. 外部接口

OTP集成含一上游APB接口，内部划分为两个独立接口。此设计确保硬件任一时刻仅处理单一上游APB访问，且采用单一PPROT安全级别。

第一个APB接口是数据接口（或数据桥）（OTPD）。其具有以下特性：

- 只读
- 连接至Synopsys设备访问端口（DAP）
- 数据接口读取始终返回32位或24位有效数据
- 数据接口地址向下取整至32位的倍数，以确保窄宽读取返回正确的字节通道
- 具有一个8 kB的窗口，支持32位ECC读取
  - 每个上游总线读取操作拆分为两次OTP访问，每次返回16位经过纠错的数据
- 具有一个8 kB的窗口，支持保护式32位ECC读取，保护读取失败时返回总线错误
  - 其功能与ECC读取窗口相同，但在访问OTP阵列前读取Synopsys启动字，首次读取若不符预期常量则返回总线错误
  - 用于提升bootrom中软件OTP读取的可靠性
- 有一个16 kB窗口，支持24位原始读取
  - 每次访问返回单个24位原始OTP行，绕过错误校正
  - 软件必须自行提供冗余（例如三重多数投票）
  - 支持位可变数据结构，如启动标志或温度计计数器；第二个APB接口为命令接口
- 提供以下两项主要功能：
  - 作为SBPI接口（Synopsys专有串行及字节并行接口总线）的桥接
    - SBPI连接至可编程主控制器（PMC），可访问DAP、DATAPATH、充电泵（IPS）及熔丝存储器（SHF）

- 允许执行任意OTP操作，包括编程
- 仅安全域可读写访问
- 提供Raspberry Pi硬件的控制寄存器
  - 寄存器根据安全/非安全及读/写权限不同，具备不同的访问权限
  - 软件锁寄存器始终对两个安全域可读

加电序列期间，从OTP读取的硬件配置信息驱动系统级控制信号，例如禁用CoreSight APs。详见第13.3.4节。

单一系统级中断输出（IRQ）为下列源产生中断：

- 因锁定导致安全读取失败
- 因锁定导致非安全读取失败
- 因锁定导致写入失败
- SBPI 标志，由 PMC 用于指示完成状态
- 当 DCTRL 设置时，数据端口访问错误
  - **USR.DCTRL**指示 SNPS AP 是否允许 SBPI 桥或数据桥访问存储器阵列；当 DAP 无法访问时，若尝试数据访问，此功能有助于调试软件

任何访问失败同时返回总线错误（**PSLVERR**）。要确定 OTP 地址是否可访问，请查询锁定表。

非安全代码无权访问中断状态寄存器。

### 13.3.3. OTP 启动振荡器

OTP 启动序列（第 13.3.4 节）由专用于 OTP 子系统的本地环形振荡器驱动。该振荡器独立于系统环形振荡器（ROSC），后者为处理器启动时提供系统时钟。

- OTP 引导振荡器是 OTP 上电状态机唯一使用的时钟。
- OTP 启动振荡器动态随机调整其自身的频率控制，以故意向时钟添加抖动。
- OTP 启动振荡器在 PSM 完成时停止，并且在 OTP 复位之前不会重新启动。
- 当 OTP 启动振荡器停止时，OTP 时钟会自动切换至 **clk\_ref**。

启动振荡器的额定频率为 12MHz。它为从 OTP 读取硬件配置提供时钟，包括配置硬件安全功能（如调试禁用和故障检测器）的关键标志（第 13.4 节）。

将该振荡器保持在 OTP 硬件子系统本地，因降低了开关时钟电容，从而减小了时钟本身的功率特征。结合频率控制的随机抖动，此举有助于防止通过功率特征分析攻击恢复 OTP 访问密钥和调试密钥，或通过对 OTP 时钟实施时序故障注入以禁用安全功能。

仅 OTP 启动振荡器默认启用 ROSC 频率随机化功能：对于后续使用系统 ROSC（第 8.3 节）的操作，必须通过编程 ROSC 控制寄存器显式启用该振荡器的此功能。晶体振荡器（XOSC）不支持频率随机化。

### 13.3.4. 上电状态机

OTP 是交换核心域电源开启状态机（第 7.4 节）中的第二项，位于处理器冷复位之后。OTP 在读取 OTP 中驻留的硬件配置之前，不释放其 **rst\_done** 信号，也不启用任何调试接口（包括第 10.10 节所述的工厂测试 JTAG）。向系统输出的 **rst\_done** 信号

PSM保持系统其余部分处于复位状态，直至OTP PSM完成，以确保在OTP内容确认之前不运行任何软件。

OTP启动序列由本地环形振荡器驱动。该振荡器专用于OTP子系统，与处理器启动时使用的主系统ROSC分开。序列如下：

1. 首先，PSM执行Synopsys启动指令。该过程包括以下步骤：
  - a. 等待电源恢复至“正常”电压值。
  - b. 读取一致性检查位置，直至硬件连续16次读取到正确数值。一致性检查使用存储于与OTP单元模拟特性相似的掩码只读存储器单元中的预定义字。
2. 读取关键标志（非ECC）：每个位在8个OTP行中冗余存储，每个标志采用八选三投票机制。
3. 通过ECC读取接口读取硬件访问密钥。
4. 读取硬件访问密钥的有效位，包括调试密钥（第3.5.9.2节）。
5. 通过原始读取接口从锁定页初始化页锁存寄存器。
6. 向系统上电状态机断言 `st_done`信号。
7. 系统复位序列继续，始于系统ROSC。

RP2350 A3为PSM增加了正确性校验和稳健性。有关这些新增功能的详细信息，请参见RP2350-E16。

## 13.4. 关键标志

关键标志启用硬件安全功能，这是RP2350安全启动实现的基本保障。OTP上电状态机在系统复位序列的早期阶段读取关键标志，先于处理器代码执行。

大多数关键标志位于主启动配置页，页1。这些标志列于OTP数据列表中的CRIT1项下。

例外的是Arm/RISC-V禁用标志，位于芯片信息页，页0。该页面在工厂编程期间被设为只读，用户无法写入CRIT0标志。

关键标志定义0为未编程值，1为已编程值。在空白设备上，所有CRIT1标志均为0。下文指定的复位值是指OTP复位施加与OTP PSM完成之间，分配给内部逻辑网络的值。例如，调试禁用标志的1复位值表示在OTP PSM运行期间调试不可访问，但之后可能可用，具体取决于从OTP存储读取的值。

- **ARM\_DISABLE**（复位：0）：强制将ARCHSEL寄存器设置为RISC-V，其优先级高于RISC-V禁用标志、安全启动启用标志或默认启动架构标志。
- **RISCV\_DISABLE**（复位：0）：强制将ARCHSEL寄存器设置为Arm，其优先级高于默认启动架构标志。
- **SECURE\_BOOT\_ENABLE**（复位：1）：启用bootrom中的启动签名校验，禁用工厂JTAG，并强制将ARCHSEL寄存器设置为Arm，其优先级高于默认启动架构标志。
- **SECURE\_DEBUG\_DISABLE**（复位：1）：禁用工厂JTAG，阻止Mem-APs对安全访问的请求，并阻止对安全处理器的停止请求。
  - 通过屏蔽其`ap_secure_en`信号，防止安全AP访问。
  - 通过屏蔽Cortex-M33的`SPIDEN`和`NSPIDEN`信号，防止安全处理器被停止。
  - 安全调试可通过OTP区块中的安全寄存器重新启用。
  - 安全调试的重新启用可通过OTP区块中的安全写1即锁寄存器禁用。
- **DEBUG\_DISABLE**（复位：1）：完全禁用Mem-APs，此外还禁用所有安全模式下已被禁用的功能。

调试禁用标志。

- **BOOT\_ARCH** (复位值: 0) : 设置ARCHSEL寄存器的复位值 (0 → Arm, 1 → RISC-V) , 前提是未被其他关键标志强制覆盖。
  - 非关键, 但由硬件读取。
- **GLITCH\_DETECTOR\_ENABLE** (复位值: 0) : 向毛刺检测器传递使能信号, 使其在任何软件运行前即可启动。
- **GLITCH\_DETECTOR\_SENS** (复位值: 0) : 配置毛刺检测电路的初始灵敏度。

关键标志通过三投八表决方式编码, 分布于连续八行OTP中。每个标志均冗余写入连续八行的相同位位置。当该位在

<sup>1</sup> 这八行中至少有三行读取为1时, 硬件即判定该标志已置位。当观察到的置位位数不超过两位时, 视为该标志已清除。

#### **i 注意**

自RP2350 A3版本起, **ARM\_DISABLE**标志不再生效, 从而消除了对已安全锁定RP2350的调试潜在解锁路径。此外, **RISCV\_DISABLE=1**与**BOOT\_ARCH=1**的组合被解码为无效状态, 芯片因此无法启动。

仅当客户RMA标志 (第13.7节) 被设置时, 才会忽略JTAG禁用。

有关关键标志影响的进一步讨论, 请参阅:

- 第3.5.9.1节, 调试禁用标志的影响
- 第3.9节, Arm/RISC-V架构选择标志的影响
- 第10.9节, 故障检测器配置标志的影响
- 第10.1.1节, 关于由**SECURE\_BOOT\_ENABLE**标志启用的BootROM安全启动支持的讨论

## 13.5. 页锁

OTP保护硬件逻辑上将OTP分为64个页 (0至63), 每页128字节, 等同于64个OTP行。

每个页面设有一组锁寄存器, 用于确定安全代码和非安全代码对该页的读写权限。锁寄存器在复位时从OTP预加载, 随后可由软件逐步修改 (即权限收紧) 。锁寄存器本身始终为全局可读。

第61至63页并非由单一组锁寄存器完整描述。这些页面存储锁初始化的元数据。更多详情请参见第13.5.4节, 本节描述由一组页面锁保护页面的较常见情况。

### 13.5.1. 锁定进度

由于硬件限制 (见第13.3.1节), 读写限制并非相互独立: 无法仅禁止对某地址的读取而不禁止写入。因此, 某页面的锁定流程如下:

0. 读写
1. 只读
2. 不可访问

锁定状态只能递增。该规则通过以下两种方式强制实施:

- 鉴于OTP的特性及编码方式, 启动时预加载至锁寄存器的OTP值不可降低。

- 锁寄存器忽略低于当前值的写入操作。

安全区和非安全区使用不同的锁定值，且可独立升级。由于非安全区域无论如何都无法直接写入OTP，因此非安全可读写与非安全只读在硬件上无区别。编码仍然有意义，因为安全软件代表非安全软件执行写入操作时，可以检查并强制执行非安全写锁。

您可以将位从任意状态重新编程至任意更高状态。锁使用2位温度计码：初始全零状态为可读写，锁通过先编程第一个位为0，再编程第二个位为1来升级。

OTP中的锁定位采用三重冗余，并通过多数投票机制确定最终值。这些位不能受到ECC保护，因为它们可能在多次编程操作中逐位发生变化。

驻留在OTP中的锁定位受其自身的安全锁级别写保护。锁页始终对所有用户开放读取权限。

安全锁寄存器可由安全代码升级，且对所有用户开放读取权限。

非安全锁寄存器可由安全或非安全代码升级，且对所有用户开放读取权限。

### 13.5.2. OTP 访问密钥

第61页包含128位密钥。每个密钥具有一个有效位：设置后，密钥对软件完全不可访问。密钥在启动时始终由硬件读取至隐藏寄存器，以便硬件在不向软件暴露密钥的情况下执行密钥比对。

某些页面可能需要特定密钥以授予相应权限。要解锁页面，用户需将其密钥写入OTP模块中的写专用寄存器。只要该寄存器中存在正确密钥，页面即保持解锁状态。若需重新锁定页面，应通过向密钥寄存器写入零值来擦除激活的密钥。

每页锁定配置指定以下内容：

- 一个读取密钥索引，范围1-7，若无读取密钥则为0
- 一个写入密钥索引，范围1-7，若无写入密钥则为0
- 无密钥状态：当软件未在密钥寄存器中输入注册密钥时，页面处于只读或不可访问状态

无密钥状态的编码如下：

- 0 表示只读（锁定级别 1）
- 1 表示不可访问（锁定级别 2）

#### 提示

配置中不存在密钥索引7。若指定密钥索引7，则保证永远不会匹配。

硬件通过比较输入密钥与当前页面的密钥配置，确定密钥锁定级别，具体如下：

1. 如果未注册任何密钥，则密钥锁定级别为0。
2. 若已注册密钥但输入密钥不匹配，则密钥锁定级别为2或1，具体取决于“无密钥状态”配置。
3. 若已注册且存在写密钥，则密钥锁定级别为0。
4. 若已注册且存在读密钥，则密钥锁定级别为1。

硬件将密钥锁定级别与当前安全域（安全/非安全）页面的锁定级别进行比较，取较高者。例如，若某页面被设为非安全只读，则任何密钥均无法将其设为非安全可写。

OTP 中存储了六个 128 位访问密钥。密钥 5 和 6 分别还作为安全调试访问密钥和

非安全调试访问密钥。有关调试密钥如何影响外部调试访问的信息，请参见第 3.5.9.2 节。

如果引导加载程序包含仅限板卡所有者安全写入的 OTP 配置，而非设备上的通用安全软件，则可能会使用 OTP 访问密钥。

### 13.5.3. OTP 中的锁定编码

页面锁定以 16 位值编码。该值以一对三重冗余字节存储，每个字节占据一个 24 位的 OTP 行。

锁定半字编码如下：

位	用途
2:0	写入密钥索引，若无写入密钥则为 0
5:3	读取密钥索引，若无读取密钥则为 0
6	无密钥状态，0=只读，1=不可访问
7	保留
9:8	安全锁状态（温度计代码 0 → 2）
11:10	非安全锁定状态（温度计代码 0 → 2）
13:12	PicoBoot 锁定状态（温度计代码 0 → 2）或在禁用 PicoBoot OTP 时的软件定义用途
15:12	保留

### 13.5.4. 特殊页面

以下页面在锁定检查中需特殊处理：

- 锁定位区域本身（第62和63页）
  - 锁定位始终对全局可读
  - 若锁定位本身允许安全代码写入，则锁定位可由安全代码写入
  - 因此，锁定位62和63被视为“备用”，因其不保护第62和63页；第63页的锁定位被重新用作RMA标志
- 硬件访问密钥页（第61页）
  - 包含OTP访问密钥和调试访问密钥
  - 每个密钥还具有有效位（*rbit*）
  - 第61页（密钥页）受第61页锁定位的所有常规保护
  - 若密钥的有效位被设置，则该密钥不可访问；反命题不一定成立

第0页，称为芯片信息页，非特殊页。Raspberry Pi在工厂测试期间，将第0页设置为只读，此前写入芯片识别和校准数值。

### 13.5.5. 空白设备权限

每个RP2350设备在制造测试阶段均编程写入部分信息。此时，还会编程写入少量硬分页锁定位：

- 第0页包含芯片信息，所有访问均为只读。
- 第1页和第2页包含启动配置和启动密钥指纹，非安全访问为只读，安全访问及启动加载程序访问为读写。
- 第62页仅包含页面62锁定位，非安全访问为只读，安全访问及启动加载程序访问为读写（作为RP2350-E28的部分解决方案）。
- 第63页包含RMA标志，非安全访问及启动加载程序访问为只读，安全访问为读写。

此空白设备上的最小默认权限集可防止某些安全模型违规情况，例如非安全代码通过用无效数据覆盖启动密钥指纹而使芯片变砖。在此语境中，术语“空白设备”指经过制造测试编程但未由用户编程任何其他OTP位的设备。

您可以在这些默认权限基础上添加额外的软锁或硬锁，但第0页除外。第0页不可硬锁定，因为安全只读权限阻止用户更改其锁字。

锁字2至61涵盖所有具有用户定义内容的页面，且保持未编程状态。在空白设备上，这些页面对所有域均完全开放访问。在启动任何非安全应用之前，应至少对所有未明确分配给非安全用途的页面施加软只读锁。具体操作为写入[SW\\_LOCK2](#)至[SW\\_LOCK61](#)。对于那些不打算进行退货维修（RMA）的设备，例如已通过板级制造测试的设备，应锁定对[RMA](#)标志的安全写入权限。

## 13.6. 误差纠正码 (ECC)

ECC保护的行以以下结构存储数据，可以通过raw别名访问：

- Bits [23:22](#)：极性位修复（BRP）标志
- Bits [21:16](#)：修正的汉明ECC码
- Bits [15:0](#)（16个最低有效位）：数据

RP2350在每个24位行的8个最高有效位中存储以下纠错数据：

- 6位修正汉明码ECC，提供单错误更正和双错误检测能力
- 2位极性位修复（BRP），支持在编程时对整行取反，以修复应为清零的单个位

写操作先编码ECC，然后编码BRP。读操作先解码BRP，再解码ECC。通过ECC数据别名（第13.1节）读取时，硬件会透明地执行纠错。使用bootrom [otp\\_access](#) API（第5.4.8.21节）进行ECC编程操作（写入）时，会自动生成ECC位。

ECC不适用于逐位突变的数据，因为ECC值来源于整个16位数据值。存储无ECC的数据时，应采用其他形式的冗余机制，如三路多数表决。

### 13.6.1. 极性位修复 (BRP)

极性比特修复(BRP)用于补偿编程时存在的单个位。

编程一行时，硬件或软件首先计算由以下部分组成的24位目标值：

- 位 [23:22](#) 的两个零
- 位 [21:16](#) 的 6 位 Hamming ECC
- 位 [15:0](#) 的 16 位数据值

编程前，OTP行应全部置零。然而，有时OTP行中已存在单个位被设置为1，可能由制造缺陷或之前的编程导致。如果OTP行中的某个位已被设置为(1)

编程前，BRP会检查目标值中对应位的状态。BRP根据目标值中对应位的状态，以两种方式之一补偿该单个位：

- 如果该位为清零（0），BRP将翻转目标行，并在位 23:22写入两个1。
- 如果该位为置位（1），BRP不会翻转目标行，保持位 23:22为两个0。

通过ECC别名读取OTP值时（第13.1节），BRP会检查位 23:22中是否为两个1。当位23和位22均为置位时，BRP会在传递至改进汉明码阶段之前翻转整行数据。

BRP允许在最初最多有一位位置的行中存储任意22位值，同时保持改进汉明码的纠错余量。制造测试期间，硬件会扫描整个OTP阵列，确保无任何行包含超过一个预置位。

### 13.6.2. 改进型汉明ECC

ECC基于存储于OTP行位15:0的数据生成六个奇偶校验位。在对行编程时，ECC生成这六个位并将其作为位 21:16包含于目标值中。该码包括：

- 一种能识别单比特错误的5位海明码
- 用于使海明码及原始16位数据检测两位错误的偶校验位

当通过ECC别名读取OTP值时（参见第13.1节），ECC会基于从OTP行读取的值重新计算六个位校验位。随后，ECC将原始六个位校验位与新计算的校验位进行异或操作。由此产生6个新位：

- 最低5位为综合症码，该码为对应数据值中每种可能比特翻转的唯一位模式
- 最高位用于区分奇数和偶数个比特翻转

若此值的所有6位均为零，说明ECC未检测到错误。若最高位为1，综合症码应指示单比特错误。ECC将翻转对应的数据位以纠正该错误。若最高位为0且综合症码非零，说明ECC检测到无法纠正的多比特错误。

您可以使用以下C代码计算5位哈明码及奇偶校验位（改编自RP2350只读存储器源代码）：

```

uint32_t even_parity(uint32_t input) {
 uint32_t rc = 0;
 while (input) {
 rc ^= input & 1;
 input >>= 1;
 }
 return rc;
}

const uint32_t otp_ecc_parity_table[6] = {
 0b0000001010110101011011,
 0b0000000011011001101101,
 0b0000001100011110001110,
 0b000000000011111110000,
 0b0000001111100000000000,
 0b0111111111111111111111
};

uint32_t s_otp_calculate_ecc(uint16_t x) {
 uint32_t p = x;
 for (uint i = 0; i < 6; ++i) {
 p |= even_parity(p & otp_ecc_parity_table[i]) << (16 + i);
 }
 return p;
}

```

}

## 13.7. 设备退役 (RMA)

退役是指当设备达到其安全生命周期终点时，销毁设备敏感内容并恢复部分测试或调试功能。OTP硬件实际上无法在未规避写保护的情况下销毁用户数据。退役功能通过**RMA**标志实现，该标志对设备进行如下修改：

- 重新启用工厂测试JTAG，该功能通常被安全启动关键标志禁用。
- 使第3页至第61页不可访问。

RMA标志不改变第0页（制造数据）、第1页和第2页（启动配置）、第61页（OTP访问密钥）以及第62页和第63页（锁定）权限。

RMA标志编码于第63页锁字的备用位。该锁字通常未被使用，因为第63页为锁定页之一；因此，该锁字不受保护。锁字分别自我保护。

与所有其他锁定字一样，第63页的锁定字受到其自身锁定的保护，这意味着它可以被硬锁定和软锁定，以防止设置RMA标志。锁定RMA标志会导致在CRIT1.SECURE\_BOOT\_ENABLE、CRIT1.DEBUG\_DISABLE或CRIT1.SECURE\_DEBUG\_DISABLE任一设置时，无法重新启用工厂JTAG接口。

这导致Raspberry Pi无法在设备退回进行故障分析时重新测试该等设备。

### ① 重要

设置RMA标志不会破坏OTP内容，仅使其无法访问。设计意图是该操作不可逆，但硬件从未达到完美。用户在对含有敏感OTP内容的设备设置RMA标志时，必须将此因素纳入其威胁模型——例如通过针对每台设备个性化OTP密钥，以防止攻击者获取密钥后造成整个类别安全破坏。

## 13.8. 镜像漏洞

RP2350 OTP用于存储启动密钥指纹及启动解密密钥。保护外部闪存存储中加密内容的能力，取决于防止只读存储器内容被未经授权或外部读取的能力。只读存储器使用 **antifuse** 位单元，该单元以电荷形式存储数据，类似于闪存位单元。它们不依赖传统熔丝单元中使用的物理结构变化。因此，它们对多种成像技术具备抗性，如光学显微镜和扫描电子显微镜。然而，antifuse 位单元可以利用一种称为**被动电压对比** (PVC) 的新型技术进行成像，该技术使用聚焦离子束 (FIB) 设备。

### PVC 白皮书

有关被动电压对比成像的更多信息，请参阅 IOActive 发布的白皮书：

<https://www.ioactive.com/wp-content/uploads/2025/01/IOActive-RP2350HackingChallenge.pdf>

此过程需进行芯片脱封，故严格要求对设备进行物理访问，且存在在无法恢复只读存储器内容的情况下损坏芯片的中等风险。

### 13.8.1. 最佳实践

以下最佳实践可最大程度降低您暴露只读存储器内容成像的风险：

- 为每台设备分配唯一密钥，避免在大量设备间共享密钥。
- 如下一节所述，使用 **chaff** 以增加成像的难度。

### 13.8.2. 干扰数据

OTP位成对出现：两个比特存储在两个晶体管的隔离栅极中，二者之间有一条公共比特线。该结构被称为比特单元。在每个64行OTP页中，第  $i$  行与第  $32+i$  行共享相同的比特单元。例如，ECC半字BOOTKEY0\_0和BOOTKEY2\_0在物理上共址。

PVC技术的具体细节使得难以区分比特单元中哪一位被设定。如果已知每对中的一个比特为零——例如，密钥存储在其他部分为空白页的底部——则可以轻易地从PVC图像读取数据。然而，当两个比特均包含未知数据时，这些尝试将被挫败。

此事实可用于在每页的上半部和下半部对数据进行冗余存储。具体如下：

- 在每行  $i$  中存储任意数据，范围从 0 至 31。
- 在每行  $32+i$  中存储这些值的24位按位取反。

此处指定的按位操作针对包含ECC位模式的整个24位原始行内容执行。

另一种方法是在行  $32+i$  中存储一个随机值，并在行  $i$  中存储该随机值与所需数据值的XOR。从功耗侧信道角度来看，此方法具有优势，因为它避免了直接从OTP读取秘密值，RP2350加密引导程序示例采用了类似的4路XOR技术。然而，对于成对噪点，推荐使用上述的按位取反技术。这与XOR技术相同，固定XOR模式为 `0xffffffff`。

## 13.9. 寄存器列表

OTP控制寄存器起始地址为 `0x40120000`（在SDK中定义为OTP\_BASE）。

表1332。  
OTP寄存器列表

偏移量	名称	说明
0x000	<a href="#">SW_LOCK0</a>	第0页的软件锁定寄存器。
0x004	<a href="#">SW_LOCK1</a>	第1页软件锁存器。
0x008	<a href="#">SW_LOCK2</a>	第2页软件锁存器。
0x00c	<a href="#">SW_LOCK3</a>	第3页软件锁存器。
0x010	<a href="#">SW_LOCK4</a>	第4页软件锁存器。
0x014	<a href="#">SW_LOCK5</a>	第5页软件锁存器。
0x018	<a href="#">SW_LOCK6</a>	第6页软件锁存器。
0x01c	<a href="#">SW_LOCK7</a>	第7页软件锁存器。
0x020	<a href="#">SW_LOCK8</a>	第8页软件锁存器。
0x024	<a href="#">SW_LOCK9</a>	第9页软件锁存器。
0x028	<a href="#">SW_LOCK10</a>	第10页软件锁存器。
0x02c	<a href="#">SW_LOCK11</a>	第11页软件锁存器。
0x030	<a href="#">SW_LOCK12</a>	第12页软件锁存器。
0x034	<a href="#">SW_LOCK13</a>	第13页软件锁存器。
0x038	<a href="#">SW_LOCK14</a>	第14页软件锁存器。
0x03c	<a href="#">SW_LOCK15</a>	第15页软件锁存器。

偏移量	名称	说明
0x040	SW_LOCK16	第16页软件锁存器。
0x044	SW_LOCK17	第17页软件锁存器。
0x048	SW_LOCK18	第18页软件锁存器。
0x04c	SW_LOCK19	第19页软件锁存寄存器。
0x050	SW_LOCK20	第20页软件锁存寄存器。
0x054	SW_LOCK21	第21页软件锁存寄存器。
0x058	SW_LOCK22	第22页软件锁存寄存器。
0x05c	SW_LOCK23	第23页软件锁存寄存器。
0x060	SW_LOCK24	第24页软件锁存寄存器。
0x064	SW_LOCK25	第25页软件锁存寄存器。
0x068	SW_LOCK26	第26页软件锁存寄存器。
0x06c	SW_LOCK27	第27页软件锁存寄存器。
0x070	SW_LOCK28	第28页软件锁存寄存器。
0x074	SW_LOCK29	第29页软件锁存寄存器。
0x078	SW_LOCK30	第30页软件锁存寄存器。
0x07c	SW_LOCK31	第31页软件锁存寄存器。
0x080	SW_LOCK32	第32页软件锁存寄存器。
0x084	SW_LOCK33	第33页软件锁存寄存器。
0x088	SW_LOCK34	第34页软件锁存寄存器。
0x08c	SW_LOCK35	第35页软件锁存寄存器。
0x090	SW_LOCK36	第36页软件锁存寄存器。
0x094	SW_LOCK37	第37页的软件锁存寄存器。
0x098	SW_LOCK38	第38页的软件锁存寄存器。
0x09c	SW_LOCK39	第39页的软件锁存寄存器。
0x0a0	SW_LOCK40	第40页的软件锁存寄存器。
0x0a4	SW_LOCK41	第41页的软件锁存寄存器。
0x0a8	SW_LOCK42	第42页的软件锁存寄存器。
0x0ac	SW_LOCK43	第43页的软件锁存寄存器。
0x0b0	SW_LOCK44	第44页的软件锁存寄存器。
0x0b4	SW_LOCK45	第45页的软件锁存寄存器。
0x0b8	SW_LOCK46	第46页的软件锁存寄存器。
0x0bc	SW_LOCK47	第47页的软件锁存寄存器。
0x0c0	SW_LOCK48	第48页的软件锁存寄存器。
0x0c4	SW_LOCK49	第49页的软件锁存寄存器。
0x0c8	SW_LOCK50	第50页的软件锁存寄存器。
0x0cc	SW_LOCK51	第51页的软件锁存寄存器。

偏移量	名称	说明
0x0d0	SW_LOCK52	第52页的软件锁存寄存器。
0x0d4	SW_LOCK53	第53页的软件锁存寄存器。
0x0d8	SW_LOCK54	第54页的软件锁存寄存器。
0x0dc	SW_LOCK55	第55页的软件锁存寄存器。
0x0e0	SW_LOCK56	第56页的软件锁存寄存器。
0x0e4	SW_LOCK57	第57页的软件锁存寄存器。
0x0e8	SW_LOCK58	第58页的软件锁存寄存器。
0x0ec	SW_LOCK59	第59页的软件锁存寄存器。
0x0f0	SW_LOCK60	第60页的软件锁存寄存器。
0x0f4	SW_LOCK61	第61页的软件锁存寄存器。
0x0f8	SW_LOCK62	第62页的软件锁存寄存器。
0x0fc	SW_LOCK63	第63页的软件锁存寄存器。
0x100	SBPI_INSTR	向SBPI接口发送指令，用于OTP熔丝编程。
0x104	SBPI_WDATA_0	SBPI写入载荷字节3..0
0x108	SBPI_WDATA_1	SBPI写入载荷字节7..4
0x10c	SBPI_WDATA_2	SBPI写入载荷字节11..8
0x110	SBPI_WDATA_3	SBPI写入载荷字节15..12
0x114	SBPI_RDATA_0	读取载荷字节3..0。读取后，寄存器中的数据将自动清零。
0x118	SBPI_RDATA_1	读取载荷字节7..4。读取后，寄存器中的数据将自动清零。
0x11c	SBPI_RDATA_2	读取载荷字节11..8。读取后，寄存器中的数据将自动清零。
0x120	SBPI_RDATA_3	读取载荷字节15..12。读取后，寄存器中的数据将自动清零。
0x124	SBPI_STATUS	
0x128	USR	APB 数据读取接口控制（用户接口）
0x12c	DBG	OTP 上电状态机调试
0x134	BIST	BIST 期间，统计至少包含一个泄漏位的地址数量
0x138	CRT_KEY_W0	密钥字 0 (位 31..0)，仅写入，读取返回 0x0
0x13c	CRT_KEY_W1	密钥字 1 (位 63..32)，仅写入，读取返回 0x0
0x140	CRT_KEY_W2	密钥字 2 (位 95..64)，仅写入，读取返回 0x0
0x144	CRT_KEY_W3	密钥字 3 (位 127..96)，仅写入，读取返回 0x0
0x148	关键	快速检查启动时读取的关键标志值
0x14c	KEY_VALID	启动时有效（已登记）的密钥

偏移量	名称	说明
0x150	DEBUGEN	启用已禁用的调试功能若 OTP 中设置了相关关键启动标志（DEBUG_DISABLE 或 SECURE_DEBUG_DISABLE），或 OTP 中标记调试密钥为有效但未通过 SWD 提供匹配密钥，则调试功能被禁用。
0x154	DEBUGEN_LOCK	写入1以锁定DEBUGEN中对应的位。该寄存器在处理器冷复位时重置。
0x158	ARCHSEL	架构选择（Arm/RISC-V），于下一次处理器复位时生效。该寄存器的默认值及其允许值受关键启动标志限制。
0x15c	ARCHSEL_STATUS	获取每个核心当前的架构选择状态。核心在软复位解除时采样ARCHSEL寄存器的当前值，同时该寄存器中相应位也会更新。
0x160	BOOTDIS	指示启动只读存储器在下一次上电时忽略擦写寄存器的启动向量（包括电源管理和看门狗）。
0x164	中断	原始中断
0x168	INTE	中断使能
0x16c	INTF	中断触发
0x170	INTS	掩码与强制后的中断状态

## OTP: SW\_LOCK0、SW\_LOCK1、...、SW\_LOCK62、SW\_LOCK63寄存器

偏移地址：0x000、0x004、...、0x0f8、0xfc

### 描述

第 N 页的软件锁定寄存器。

复位时，锁定状态由OTP锁定页初始化。该寄存器可写入，以在下一次复位前进一步更新每页的锁定状态，也可读取以检查某页的当前锁定状态。

表1333。  
SW\_LOCK0,  
SW\_LOCK1,...  
SW\_LOCK62,  
SW\_LOCK63寄存器

位	描述	类型	复位
31:4	保留。	-	-
3:2	<b>NSEC</b> ：非安全锁状态。写入操作将与当前值进行按位或运算。	读写	-
	枚举值：		
	0x0 → 读写		
	0x1 → 只读		
	0x3 → 不可访问		
1:0	<b>SEC</b> ：安全锁状态。写入操作将与当前值进行按位或运算。该字段对非安全代码为只读。	读写	-
	枚举值：		
	0x0 → 读写		
	0x1 → 只读		
	0x3 → 不可访问		

## OTP: SBPI\_INSTR寄存器

偏移量: 0x100

### 描述

向SBPI接口发送指令，用于OTP熔丝编程。

表1334。  
SBPI\_INSTR寄存器

位	描述	类型	复位
31	保留。	-	-
30	<b>EXEC</b> : 执行指令	SC	0x0
29	<b>IS_WR</b> : 载荷类型为写	读写	0x0
28	<b>HAS_PAYLOAD</b> : 指令包含载荷（写入或读取的数据）	读写	0x0
27:24	<b>PAYOUT_SIZE_M1</b> : 指令载荷大小（字节）减一	读写	0x0
23:16	<b>TARGET</b> : 指令目标，可为PMC（0x3a）或DAP（0x02）	读写	0x00
15:8	<b>CMD</b>	读写	0x00
7:0	<b>SHORT_WDATA</b> : 仅在payload_size_m1=0时使用的wdata	读写	0x00

## OTP: SBPI\_WDATA\_0寄存器

偏移量: 0x104

表1335  
SBPI\_WDATA\_0  
寄存器

位	描述	类型	复位
31:0	SBPI写入载荷字节3..0	读写	0x00000000

## OTP: SBPI\_WDATA\_1寄存器

偏移量: 0x108

表1336  
SBPI\_WDATA\_1  
寄存器

位	描述	类型	复位
31:0	SBPI写入载荷字节7..4	读写	0x00000000

## OTP: SBPI\_WDATA\_2寄存器

偏移量: 0x10c

表1337  
SBPI\_WDATA\_2  
寄存器

位	描述	类型	复位
31:0	SBPI写入载荷字节11..8	读写	0x00000000

## OTP: SBPI\_WDATA\_3寄存器

偏移量: 0x110

表1338  
SBPI\_WDATA\_3  
寄存器

位	描述	类型	复位
31:0	SBPI写入载荷字节15..12	读写	0x00000000

## OTP: SBPI\_RDATA\_0寄存器

偏移量: 0x114

表1339  
SBPI\_RDATA\_0  
寄存器

位	描述	类型	复位
31:0	读取载荷字节3..0。读取后，寄存器中的数据将自动清零。	只读	0x00000000

## OTP: SBPI\_RDATA\_1 寄存器

偏移: 0x118

表1340  
SBPI\_RDATA\_1  
寄存器

位	描述	类型	复位
31:0	读取载荷字节7..4。读取后，寄存器中的数据将自动清零。	只读	0x00000000

## OTP: SBPI\_RDATA\_2 寄存器

偏移: 0x11c

表 1341。  
SBPI\_RDATA\_2  
寄存器

位	描述	类型	复位
31:0	读取载荷字节11..8。读取后，寄存器中的数据将自动清零。	只读	0x00000000

## OTP: SBPI\_RDATA\_3 寄存器

偏移: 0x120

表 1342。  
SBPI\_RDATA\_3  
寄存器

位	描述	类型	复位
31:0	读取载荷字节15..12。读取后，寄存器中的数据将自动清零。	只读	0x00000000

## OTP: SBPI\_STATUS 寄存器

偏移: 0x124

表 1343。  
SBPI\_STATUS 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>MISO:</b> SBPI MISO (主入-从出) : 来自 SBPI 的响应	只读	-
15:13	保留。	-	-
12	<b>FLAG:</b> SBPI 标志	只读	-
11:9	保留。	-	-
8	<b>INSTR_MISS:</b> 上一条指令缺失 (被丢弃)，因前一条指令尚未执行完毕	WC	0x0
7:5	保留。	-	-
4	<b>INSTR_DONE:</b> 上一条指令完成	WC	0x0
3:1	保留。	-	-
0	<b>RDATA_VLD:</b> 读命令已返回数据	WC	0x0

## OTP: USR 寄存器

偏移: 0x128

**描述**

APB 数据读取接口控制（用户接口）

表 1344。USR 寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>PD</b> : 断电；1 表示禁用电流参考。必须为0才能从OTP读取数据。	读写	0x0
3:1	保留。	-	-
0	<b>DCTRL</b> : 1表示启用用户界面；0表示禁用用户界面（启用SBPI）。  执行任何SBPI访问（如编程OTP）前，必须清除此位。在此期间，APB数据读取接口（用户界面）不可用，若尝试读取则会返回总线错误。	读写	0x1

**OTP：DBG寄存器**

偏移量：0x12c

**描述**

OTP 上电状态机调试

表 1345。DBG 寄存器

位	描述	类型	复位
31:13	保留。	-	-
12	<b>CUSTOMER_RMA_FLAG</b> : 芯片处于RMA模式	只读	-
11:8	保留。	-	-
7:4	<b>PSM_STATE</b> : 监控PSM有限状态机的状态	只读	-
3	<b>ROSC_UP</b> : 环形振荡器已启动运行	只读	-
2	<b>ROSC_UP_SEEN</b> : 已检测到环形振荡器运行状态	WC	0x0
1	<b>BOOT_DONE</b> : PSM启动完成状态标志	只读	-
0	<b>PSM_DONE</b> : PSM完成状态标志	只读	-

**OTP：BIST寄存器**

偏移量：0x134

**描述**

BIST 期间，统计至少包含一个泄漏位的地址数量

表 1346。BIST 寄存器

位	描述	类型	复位
31	保留。	-	-
30	<b>CNT_FAIL</b> : 当地址位置计数中泄漏位数超过cnt_max时置位该标志	只读	-
29	<b>CNT_CLR</b> : 使用前清零计数器	SC	0x0
28	<b>CNT_ENA</b> : 在启动BIST功能前启用计数器	读写	0x0
27:16	<b>CNT_MAX</b> : 若泄漏位置数量超过此值，cnt_fail标志将被设置	读写	0xffff
15:13	保留。	-	-

位	描述	类型	复位
12:0	CNT：至少含一个泄漏位的位置数量。注：仅当BIST在未启用修复选项时启动，该计数有效。	只读	-

## OTP: CRT\_KEY\_W0寄存器

偏移: 0x138

表1347。  
CRT\_KEY\_W0寄存器

位	描述	类型	复位
31:0	密钥字 0 (位 31..0) , 仅写入, 读取返回 0x0	WO	0x00000000

## OTP: CRT\_KEY\_W1寄存器

偏移: 0x13C

表1348。  
CRT\_KEY\_W1寄存器

位	描述	类型	复位
31:0	密钥字 1 (位 63..32) , 仅写入, 读取返回 0x0	WO	0x00000000

## OTP: CRT\_KEY\_W2寄存器

偏移: 0x140

表1349。  
CRT\_KEY\_W2寄存器

位	描述	类型	复位
31:0	密钥字 2 (位 95..64) , 仅写入, 读取返回 0x0	WO	0x00000000

## OTP: CRT\_KEY\_W3寄存器

偏移: 0x144

表1350。  
CRT\_KEY\_W3寄存器

位	描述	类型	复位
31:0	密钥字 3 (位 127..96) , 仅写入, 读取返回 0x0	WO	0x00000000

## OTP: CRITICAL寄存器

偏移: 0x148

### 描述

快速检查启动时读取的关键标志值

表1351. 关键  
寄存器

位	描述	类型	复位
31:18	保留。	-	-
17	RISCV_DISABLE	只读	0x0
16	ARM_DISABLE	只读	0x0
15:7	保留。	-	-
6:5	GLITCH_DETECTOR_SENS	只读	0x0
4	GLITCH_DETECTOR_ENABLE	只读	0x0
3	DEFAULT_ARCHSEL	只读	0x0
2	DEBUG_DISABLE	只读	0x0
1	SECURE_DEBUG_DISABLE	只读	0x0
0	SECURE_BOOT_ENABLE	只读	0x0

## OTP: KEY\_VALID 寄存器

偏移: 0x14c

表1352.  
KEY\_VALID寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	启动时有效（已登记）的密钥	只读	0x00

## OTP: DEBUGEN 寄存器

偏移: 0x150

### 描述

启用已禁用的调试功能若 OTP 中设置了相关关键启动标志（DEBUG\_DISABLE 或 SECURE\_DEBUG\_DISABLE），或 OTP 中标记调试密钥为有效但未通过 SWD 提供匹配密钥，则调试功能被禁用。

具体说明：

- DEBUG\_DISABLE 标志禁用所有调试功能。该标志可通过设置此寄存器的所有位完全覆盖。
- SECURE\_DEBUG\_DISABLE 标志禁用安全处理器调试。该标志可通过设置此寄存器的 PROC0\_SECURE 和 PROC1\_SECURE 位完全覆盖。
- 如果已注册单个调试密钥，且 SWD 未提供匹配的密钥值，则所有调试功能均被禁用。该标志可通过设置此寄存器的所有位完全覆盖。
- 如果已注册两个调试密钥，且通过 SWD 提供了非安全密钥（密钥 6）的值，则安全处理器调试被禁用。该标志可通过设置此寄存器的 PROC0\_SECURE 和 PROC1\_SECURE 位完全覆盖。
- 如果已注册两个调试密钥，且通过 SWD 提供了安全密钥（密钥 5）的值，则密钥机制不禁用任何调试功能。然而，请注意，在此情况下关键启动标志仍可能禁用调试功能。

表 1353. DEBUGEN  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>MISC:</b> 启用其他调试组件。具体包括CTI及用于访问RISC-V调试模块的APB-AP。  若设置了任一调试禁用关键标志，或至少注册了一把调试密钥且未通过SWD提供这些已注册密钥中安全级别最低的密钥值，则这些组件默认被禁用。	读写	0x0
7:4	保留。	-	-
3	<b>PROC1_SECURE:</b> 允许核心1的Mem-AP生成安全访问（前提是已启用此项）。同时启用核心1的安全调试（SPIDEN和SPNIDEN）。  若设置了安全调试禁用关键标志，或至少注册了一把调试密钥且尚未通过SWD提供这些已注册密钥中安全级别最高的密钥值，则核心1的安全调试默认被禁用。	读写	0x0

位	描述	类型	复位
2	<b>PROC1</b> : 若核心1的Mem-AP当前处于禁用状态，则启用该功能。  若设置了任一调试禁用关键标志，或至少注册了一个调试密钥且这些已注册密钥中安全性最低的密钥值未通过SWD提供，则Mem-AP默认被禁用。	读写	0x0
1	<b>PROC0_SECURE</b> : 允许核心0的Mem-AP生成安全访问，前提是其已启用。同时启用核心0的安全调试（SPIDEN和SPNIDEN）。  若设置了安全调试禁用关键标志，或至少注册了一个调试密钥且这些已注册密钥中安全性最高的密钥值尚未通过SWD提供，则核心0的安全调试默认被禁用。  另请注意，当核心切换至RISC-V模式（通过设置ARCHSEL位并执行核心热复位）时，核心的Mem-AP无条件被禁用。	读写	0x0
0	<b>PROC0</b> : 若当前禁用，则启用核心0的Mem-AP。  若设置了任一调试禁用关键标志，或至少注册了一个调试密钥且这些已注册密钥中安全性最低的密钥值未通过SWD提供，则Mem-AP默认被禁用。  另请注意，当核心切换至RISC-V模式（通过设置ARCHSEL位并执行核心热复位）时，核心的Mem-AP无条件被禁用。	读写	0x0

## OTP: DEBUGEN\_LOCK 寄存器

偏移量: 0x154

### 描述

写入1以锁定DEBUGEN中对应的位。该寄存器在处理器冷复位时重置。

表1354。  
DEBUGEN\_LOCK  
寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>MISC</b> : 写入1以锁定DEBUGEN的MISC位。设置后不可清除。	读写	0x0
7:4	保留。	-	-
3	<b>PROC1_SECURE</b> : 写入1以锁定DEBUGEN的PROC1_SECURE位。设置后不可清除。	读写	0x0
2	<b>PROC1</b> : 写入1以锁定DEBUGEN的PROC1位。设置后不可清除。	读写	0x0
1	<b>PROC0_SECURE</b> : 写入1以锁定DEBUGEN的PROC0_SECURE位。设置后不可清除。	读写	0x0
0	<b>PROC0</b> : 写入1以锁定DEBUGEN的PROC0位。设置后不可清除。	读写	0x0

## OTP: ARCHSEL 寄存器

偏移: 0x158

### 描述

架构选择（Arm/RISC-V）。该寄存器的默认值及其允许值受关键启动标志限制。

该寄存器由切换核心电源域的最早复位触发复位（发生于处理器冷复位之前）。

核心在热复位时采样架构选择信号。热复位可能由系统电源上电状态机、看门狗定时器、Arm SYSRESETREQ或RISC-V hartresetreq触发。

请注意，当Arm内核被取消选择时，其冷复位域亦将保持复位状态，特别是SYSRESETREQ位在内核取消选择后将无法访问。另请注意，RISC-V内核无冷复位域，因为其对应控制位于调试模块内。

表1355。ARCHSEL  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>CORE1</b> ：选择核心1的架构。	读写	0x0
	枚举值：		
	0x0 → ARM：切换核心1至Arm (Cortex-M33)		
	0x1 → RISCV：切换核心1至RISC-V (Hazard3)		
0	<b>CORE0</b> ：选择核心0的架构。	读写	0x0
	枚举值：		
	0x0 → ARM：切换核心0至Arm (Cortex-M33)		
	0x1 → RISCV：切换核心0至RISC-V (Hazard3)		

## OTP：ARCHSEL\_STATUS寄存器

偏移量：0x15c

### 描述

获取每个核心当前的架构选择状态。核心在软复位解除时采样ARCHSEL寄存器的当前值，同时该寄存器中相应位也会更新。

表1356。  
ARCHSEL\_STATUS  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>CORE1</b> ：核心1的当前架构。处理器热复位时更新。	只读	0x0
	枚举值：		
	0x0 → ARM：核心1当前为Arm (Cortex-M33)		
	0x1 → RISCV：核心1当前为RISC-V (Hazard3)		
0	<b>CORE0</b> ：核心0的当前架构。处理器热复位时更新。	只读	0x0
	枚举值：		
	0x0 → ARM：核心0当前为Arm (Cortex-M33)		
	0x1 → RISCV：核心0当前为RISC-V (Hazard3)		

## OTP：BOOTDIS寄存器

偏移：0x160

### 描述

指示启动只读存储器在下一次上电时忽略擦写寄存器的启动向量（包括电源管理和看门狗）。

如果早期引导阶段已对某些OTP页面进行软锁定以保护其内容免受后续阶段影响，则存在安全代码在后续阶段通过执行看门狗复位以重置OTP进而解锁这些页面的风险。

此寄存器可用于确保引导加载程序在下次上电时正常运行，防止安全代码在

在其解锁状态下，后续阶段访问OTP。

应配合电源管理器BOOTDIS寄存器使用。

表1357. BOOTDIS  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>NEXT</b> : 该标志写入操作总是与其当前内容做按位或（OR）操作。软件可以设置该位，但不能清除此位。  当核心断电时，BOOTDIS_NEXT位会按位或（OR）进BOOTDIS_NOW位。 同时，BOOTDIS_NEXT位会被清零。设置此位表示下一次核心断电后将忽略启动暂存寄存器。  该标志应由已对OTP页进行软锁定的早期启动阶段设置，以防止后续阶段通过看门狗复位将其解锁。	读写	0x0
0	<b>NOW</b> : 当核心断电时，BOOTDIS_NEXT的当前值会按位或运算合并到BOOTDIS_NOW，随后BOOTDIS_NEXT被清零。  bootrom在读取启动暂存寄存器前会检查该标志。若标志被设置，bootrom将其清除并忽略BOOT寄存器内容。此举防止安全软件在引导加载程序软锁包含敏感数据的OTP页面之前篡改引导路径。	WC	0x0

## OTP: INTR寄存器

偏移: 0x164

### 描述

原始中断

表1358. INTR  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>APB_RD_NSEC_FAIL</b>	WC	0x0
3	<b>APB_RD_SEC_FAIL</b>	WC	0x0
2	<b>APB_DCTRL_FAIL</b>	WC	0x0
1	<b>SBPI_WR_FAIL</b>	WC	0x0
0	<b>SBPI_FLAG_N</b>	只读	0x0

## OTP: INTE寄存器

偏移: 0x168

### 描述

中断使能

表1359. INTE  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>APB_RD_NSEC_FAIL</b>	读写	0x0
3	<b>APB_RD_SEC_FAIL</b>	读写	0x0
2	<b>APB_DCTRL_FAIL</b>	读写	0x0

位	描述	类型	复位
1	<b>SBPI_WR_FAIL</b>	读写	0x0
0	<b>SBPI_FLAG_N</b>	读写	0x0

## OTP: INTF寄存器

偏移: 0x16c

### 描述

中断强制

表1360. INTF  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>APB_RD_NSEC_FAIL</b>	读写	0x0
3	<b>APB_RD_SEC_FAIL</b>	读写	0x0
2	<b>APB_DCTRL_FAIL</b>	读写	0x0
1	<b>SBPI_WR_FAIL</b>	读写	0x0
0	<b>SBPI_FLAG_N</b>	读写	0x0

## OTP: INTS寄存器

偏移: 0x170

### 描述

掩码与强制后的中断状态

表1361. INTS  
寄存器

位	描述	类型	复位
31:5	保留。	-	-
4	<b>APB_RD_NSEC_FAIL</b>	只读	0x0
3	<b>APB_RD_SEC_FAIL</b>	只读	0x0
2	<b>APB_DCTRL_FAIL</b>	只读	0x0
1	<b>SBPI_WR_FAIL</b>	只读	0x0
0	<b>SBPI_FLAG_N</b>	只读	0x0

## 13.10. 预定义OTP数据位置

本节列出了由硬件（尤其是OTP电源开启状态机）、bootrom或两者共同使用的OTP位置。本列表适用于RP2350 硅片A2 版本。

OTP位置按行号列出，而非按地址。通过ECC别名读取时，OTP行在系统地址空间中间隔两个字节；通过原始别名读取时，OTP行间隔四个字节。因此，直接从软件读取OTP内容时，应根据情况将此处提供的行号乘以二或四。bootrom提供的OTP API直接使用OTP行号，故通过这些API访问OTP时无须进行行号到字节地址的转换。

对于普通（非保护）读取，可以从[OTP\\_DATA\\_BASE \(0x40130000\)](#)访问经过纠错的数据内容，从[OTP\\_DATA\\_RAW\\_BASE \(0x40134000\)](#)访问原始数据内容。下列寄存器列表指示指定的OTP行是否包含经过纠错的内容。OTP绝不在同一行中混合存储经过纠错和未经纠错的内容。

所有预定义的数据字段均采用一定形式的冗余。当ECC不可用时，例如某些位置预期在不同时刻编程单个位，则改用三次投票多数决法。唯一例外是CRIT0和CRIT1中的关键硬件标志。对此类标志采用三选八投票编码：当八个位冗余位置中至少三个位被置位时，即视该标志为已设置。

每行的描述均指明其冗余类型。

第3页至第60页（即行 `0x0c0` 至 `0xf3f`）可供任意用户内容使用，例如驻留OTP的引导加载程序，Raspberry Pi在可能情况下将避免将这些位置分配给bootrom配置。这是总计7424字节的ECC保护内容。

如果禁用安全启动，页面2（行 `0x080` 至 `0x0bf`）亦可用于用户内容。若启用安全启动且注册的启动密钥指纹少于四个，则页面部分可用。这是额外的128字节ECC内容，可能供用户使用。

页面0、1及61至63预留给Raspberry Pi未来使用。软件应避免在这些区域分配内容，即便当前这些区域在本数据列表中尚无明确定义的用途。

表1362。 *OTP\_D*  
ATA寄存器列表

偏移量	名称	说明
0x000	<b>CHIPID0</b>	公共设备ID的第15至0位。（ECC）  CHIPID0至CHIPID3行包含该芯片的64位随机标识符，可通过USB引导加载程序PICOBOOT接口或get_sys_info只读存储器API读取。  随机位数使得出现重复ID的可能性极低：例如，在1亿台设备中，99.97%的概率不存在重复ID。该概率被估计低于顺序随机ID分配过程中的工艺错误概率，实际应用中CHIPID可视为唯一标识。
0x001	<b>CHIPID1</b>	公共设备ID（ECC）的第31至16位
0x002	<b>CHIPID2</b>	公共设备ID（ECC）的第47至32位
0x003	<b>CHIPID3</b>	公共设备ID（ECC）的第63至48位
0x004	<b>RANDID0</b>	私有每设备随机数（ECC）的第15至0位  RANDID0至7行组成了在设备测试期间生成的128位随机数。  该ID不会通过USB PICOBOOT的GET_INFO命令或ROMget_sys_info()API暴露。但请注意，USB PICOBOOT OTP访问点可以读取第0页的全部内容，因此除非  通过BOOT_FLAGS0中的DISABLE_BOOTSEL_USB_PICOBOOT_IFC标志禁用USB PICOBOOT接口，否则该值不具备实质性的隐私性。
0x005	<b>RANDID1</b>	私有每设备随机数（ECC）的第31至16位
0x006	<b>RANDID2</b>	私有每设备随机数的第47至32位（ECC）
0x007	<b>RANDID3</b>	私有每设备随机数的第63至48位（ECC）
0x008	<b>RANDID4</b>	私有每设备随机数的第79至64位（ECC）
0x009	<b>RANDID5</b>	私有每设备随机数的第95至80位（ECC）
0x00a	<b>RANDID6</b>	私有每设备随机数的第111至96位（ECC）
0x00b	<b>RANDID7</b>	私有每设备随机数的第127至112位（ECC）

偏移量	名称	说明
0x010	ROSC_CALIB	制造过程中测得的环形振荡器频率，单位kHz (ECC)  该测量在室温下，于ROSC配置寄存器处于复位状态时，于1.1 V电压下进行。
0x011	LPOSC_CALIB	制造过程中测得的低功耗振荡器频率，单位为Hz (ECC)  该测量在室温下，于LPOSC微调寄存器处于复位状态时，于1.1 V电压下进行。
0x018	NUM_GPIOS	主用户GPIO数量 (bank 0)。QFN80封装中应为48，QFN60封装中应为30。 (ECC)
0x036	INFO_CRC0	OTP地址0x00至0x6b的CRC32低16位 (多项式0x4c11db7，输入反射，输出反射，种子全为1，最终异或全为1) (ECC)
0x037	INFO_CRC1	OTP地址0x00至0x6b的CRC32高16位 (ECC)
0x038	CRIT0	第0页关键启动标志 (RBIT-8)
0x039	CRIT0_R1	CRIT0 的冗余副本
0x03a	CRIT0_R2	CRIT0 的冗余副本
0x03b	CRIT0_R3	CRIT0 的冗余副本
0x03c	CRIT0_R4	CRIT0 的冗余副本
0x03d	CRIT0_R5	CRIT0 的冗余副本
0x03e	CRIT0_R6	CRIT0 的冗余副本
0x03f	CRIT0_R7	CRIT0 的冗余副本
0x040	CRIT1	第1页关键启动标志 (RBIT-8)
0x041	CRIT1_R1	CRIT1 的冗余副本
0x042	CRIT1_R2	CRIT1 的冗余副本
0x043	CRIT1_R3	CRIT1 的冗余副本
0x044	CRIT1_R4	CRIT1 的冗余副本
0x045	CRIT1_R5	CRIT1 的冗余副本
0x046	CRIT1_R6	CRIT1 的冗余副本
0x047	CRIT1_R7	CRIT1 的冗余副本
0x048	BOOT_FLAGS0	禁用/启用 RP2350 只读存储器中的启动路径/功能。 禁用操作始终优先于启用操作。当OTP中存在必须有效的其他配置时，提供启用功能。 (RBIT-3)
0x049	BOOT_FLAGS0_R1	BOOT_FLAGS0的冗余副本
0x04a	BOOT_FLAGS0_R2	BOOT_FLAGS0的冗余副本
0x04b	BOOT_FLAGS1	禁用/启用 RP2350 只读存储器中的启动路径/功能。 禁用操作始终优先于启用操作。当OTP中存在必须有效的其他配置时，提供启用功能。 (RBIT-3)

偏移量	名称	说明
0x04c	BOOT_FLAGS1_R1	BOOT_FLAGS1的冗余副本
0x04d	BOOT_FLAGS1_R2	BOOT_FLAGS1的冗余副本
0x04e	DEFAULT_BOOT_VERSION0	默认启动版本温度计计数器，位23:0 (RBIT-3)
0x04f	DEFAULT_BOOT_VERSION0_R1	DEFAULT_BOOT_VERSION0的冗余副本
0x050	DEFAULT_BOOT_VERSION0_R2	DEFAULT_BOOT_VERSION0的冗余副本
0x051	DEFAULT_BOOT_VERSION1	默认启动版本温度计计数器，位47:24 (RBIT-3)
0x052	DEFAULT_BOOT_VERSION1_R1	DEFAULT_BOOT_VERSION1的冗余副本
0x053	DEFAULT_BOOT_VERSION1_R2	DEFAULT_BOOT_VERSION1的冗余副本
0x054	FLASH_DEVINFO	存储有关外部闪存设备的信息。 (ECC)  假定在以下条件下有效 BOOT_FLAGS0_FLASH_DEVINFO_ENABLE 已设置。
0x055	FLASH_PARTITION_SLOT_SIZE	闪存起始分区表槽0与槽1之间的间隙 (默认大小为4096字节) (ECC)，由 BOOT_FLAGS 中的 OVERRIDE_FLASH_PARTITION_SLOT_SIZE 位启用。 大小为 4096 * (值 + 1)
0x056	BOOTSEL_LED_CFG	LED状态的引脚配置，供USB引导加载程序使用。 (ECC) 如果设置了 BOOT_FLAGS0_ENABLE_BOOTSEL_LED，则该配置必须有效。
0x057	BOOTSEL_PLL_CFG	BOOTSEL模式的可选PLL配置。 (ECC)
0x058	BOOTSEL_XOSC_CFG	USB引导加载程序的非默认晶体振荡器配置。 (ECC)
0x059	USB_BOOT_FLAGS	USB引导特定功能标志 (RBIT-3)
0x05a	USB_BOOT_FLAGS_R1	USB_BOOT_FLAGS的冗余副本
0x05b	USB_BOOT_FLAGS_R2	USB_BOOT_FLAGS的冗余副本
0x05c	USB_WHITE_LABEL_ADDR	OTP中USB_WHITE_LABEL结构的行索引 (ECC)
0x05e	OTPBOOT_SRC	OTP引导映像的OTP起始行。 (ECC)
0x05f	OTPBOOT_LEN	OTP 启动映像的行长度。 (ECC)
0x060	OTPBOOT_DST0	OTP 启动映像加载地址 (及入口点) 位 15:0。 (ECC)
0x061	OTPBOOT_DST1	OTP 启动映像加载地址 (及入口点) 位 31:16。 (ECC)
0x080	BOOTKEY0_0	启动密钥 0 的 SHA-256 哈希值位 15:0。 (ECC)
0x081	BOOTKEY0_1	启动密钥 0 的 SHA-256 哈希值位 31:16。 (ECC)
0x082	BOOTKEY0_2	启动密钥 0 的 SHA-256 哈希值位 47:32。 (ECC)
0x083	BOOTKEY0_3	启动密钥 0 的 SHA-256 哈希值位 63:48。 (ECC)
0x084	BOOTKEY0_4	启动密钥 0 的 SHA-256 哈希值位 79:64。 (ECC)
0x085	BOOTKEY0_5	启动密钥 0 的 SHA-256 哈希值位 95:80。 (ECC)
0x086	BOOTKEY0_6	启动密钥 0 的 SHA-256 哈希值位 111:96。 (ECC)
0x087	BOOTKEY0_7	引导密钥0 (ECC) SHA-256哈希值的第127至112位

偏移量	名称	说明
0x088	BOOTKEY0_8	引导密钥0 (ECC) SHA-256哈希值的第143至128位
0x089	BOOTKEY0_9	引导密钥0 (ECC) SHA-256哈希值的第159至144位
0x08a	BOOTKEY0_10	引导密钥0 (ECC) SHA-256哈希值的第175至160位
0x08b	BOOTKEY0_11	引导密钥0 (ECC) SHA-256哈希值的第191至176位
0x08c	BOOTKEY0_12	引导密钥0 (ECC) SHA-256哈希值的第207至192位
0x08d	BOOTKEY0_13	引导密钥0 (ECC) SHA-256哈希值的第223至208位
0x08e	BOOTKEY0_14	引导密钥0 (ECC) SHA-256哈希值的第239至224位
0x08f	BOOTKEY0_15	引导密钥0 (ECC) SHA-256哈希值的第255至240位
0x090	BOOTKEY1_0	启动密钥1的SHA-256哈希值的第15至0位 (ECC)
0x091	BOOTKEY1_1	启动密钥1的SHA-256哈希值的第31至16位 (ECC)
0x092	BOOTKEY1_2	启动密钥1的SHA-256哈希值的第47至32位 (ECC)
0x093	BOOTKEY1_3	启动密钥1的SHA-256哈希值的第63至48位 (ECC)
0x094	BOOTKEY1_4	启动密钥1的SHA-256哈希值的第79至64位 (ECC)
0x095	BOOTKEY1_5	启动密钥1的SHA-256哈希值的第95至80位 (ECC)
0x096	BOOTKEY1_6	启动密钥1的SHA-256哈希值的第111至96位 (ECC)
0x097	BOOTKEY1_7	启动密钥1的SHA-256哈希值的第127至112位 (ECC)
0x098	BOOTKEY1_8	启动密钥1的SHA-256哈希值的第143至128位 (ECC)
0x099	BOOTKEY1_9	启动密钥1的SHA-256哈希值的第159至144位 (ECC)
0x09a	BOOTKEY1_10	启动密钥1 (ECC) 的SHA-256哈希第175至160位
0x09b	BOOTKEY1_11	启动密钥1 (ECC) 的SHA-256哈希第191至176位
0x09c	BOOTKEY1_12	启动密钥1 (ECC) 的SHA-256哈希第207至192位
0x09d	BOOTKEY1_13	启动密钥1 (ECC) 的SHA-256哈希第223至208位
0x09e	BOOTKEY1_14	启动密钥1 (ECC) 的SHA-256哈希第239至224位
0x09f	BOOTKEY1_15	启动密钥1 (ECC) 的SHA-256哈希第255至240位
0x0a0	BOOTKEY2_0	启动密钥2 (ECC) 的SHA-256哈希第15至0位
0x0a1	BOOTKEY2_1	启动密钥2 (ECC) 的SHA-256哈希第31至16位
0x0a2	BOOTKEY2_2	启动密钥2 (ECC) 的SHA-256哈希第47至32位
0x0a3	BOOTKEY2_3	启动密钥2 (ECC) SHA-256哈希的第63至48位
0x0a4	BOOTKEY2_4	启动密钥2 (ECC) SHA-256哈希的第79至64位
0x0a5	BOOTKEY2_5	启动密钥2 (ECC) SHA-256哈希的第95至80位
0x0a6	BOOTKEY2_6	启动密钥2 (ECC) SHA-256哈希的第111至96位
0x0a7	BOOTKEY2_7	启动密钥2 (ECC) SHA-256哈希的第127至112位
0x0a8	BOOTKEY2_8	启动密钥2 (ECC) SHA-256哈希的第143至128位
0x0a9	BOOTKEY2_9	启动密钥2 (ECC) SHA-256哈希的第159至144位
0x0aa	BOOTKEY2_10	启动密钥2 (ECC) SHA-256哈希的第175至160位
0x0ab	BOOTKEY2_11	启动密钥2 (ECC) SHA-256哈希的第191至176位

偏移量	名称	说明
0x0ac	BOOTKEY2_12	引导密钥 2 (ECC) 的 SHA-256 哈希值的第 207 至 192 位
0x0ad	BOOTKEY2_13	引导密钥 2 (ECC) 的 SHA-256 哈希值的第 223 至 208 位
0x0ae	BOOTKEY2_14	引导密钥 2 (ECC) 的 SHA-256 哈希值的第 239 至 224 位
0x0af	BOOTKEY2_15	引导密钥 2 (ECC) 的 SHA-256 哈希值的第 255 至 240 位
0x0b0	BOOTKEY3_0	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 15 至 0 位
0x0b1	BOOTKEY3_1	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 31 至 16 位
0x0b2	BOOTKEY3_2	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 47 至 32 位
0x0b3	BOOTKEY3_3	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 63 至 48 位
0x0b4	BOOTKEY3_4	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 79 至 64 位
0x0b5	BOOTKEY3_5	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 95 至 80 位
0x0b6	BOOTKEY3_6	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 111 至 96 位
0x0b7	BOOTKEY3_7	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 127 至 112 位
0x0b8	BOOTKEY3_8	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 143 至 128 位
0x0b9	BOOTKEY3_9	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 159 至 144 位
0x0ba	BOOTKEY3_10	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 175 至 160 位
0x0bb	BOOTKEY3_11	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 191 至 176 位
0x0bc	BOOTKEY3_12	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 207 至 192 位
0x0bd	BOOTKEY3_13	引导密钥 3 (ECC) 的 SHA-256 哈希值的第 223 至 208 位
0x0be	BOOTKEY3_14	启动密钥 3 (ECC) SHA-256 哈希值的第 239 至 224 位
0x0bf	BOOTKEY3_15	启动密钥 3 (ECC) SHA-256 哈希值的第 255 至 240 位
0xf48	KEY1_0	OTP 访问密钥 1 (ECC) 的第 15 至 0 位
0xf49	KEY1_1	OTP 访问密钥 1 (ECC) 的第 31 至 16 位
0xf4a	KEY1_2	OTP 访问密钥 1 (ECC) 的第 47 至 32 位
0xf4b	KEY1_3	OTP 访问密钥 1 (ECC) 的第 63 至 48 位
0xf4c	KEY1_4	OTP 访问密钥 1 (ECC) 的第 79 至 64 位
0xf4d	KEY1_5	OTP 访问密钥 1 (ECC) 的第 95 至 80 位
0xf4e	KEY1_6	OTP 访问密钥 1 (ECC) 的第 111 至 96 位
0xf4f	KEY1_7	OTP 访问密钥 1 (ECC) 的第 127 至 112 位
0xf50	KEY2_0	OTP 访问密钥 2 的第 15 至 0 (ECC)
0xf51	KEY2_1	OTP 访问密钥 2 的第 31 至 16 (ECC)
0xf52	KEY2_2	OTP 访问密钥 2 的第 47 至 32 (ECC)
0xf53	KEY2_3	OTP 访问密钥 2 的第 63 至 48 (ECC)
0xf54	KEY2_4	OTP 访问密钥 2 的第 79 至 64 (ECC)
0xf55	KEY2_5	OTP 访问密钥 2 的第 95 至 80 (ECC)
0xf56	KEY2_6	OTP 访问密钥 2 的第 111 至 96 (ECC)
0xf57	KEY2_7	OTP 访问密钥 2 的第 127 至 112 (ECC)

偏移量	名称	说明
0xf58	KEY3_0	OTP访问密钥3（ECC）的第15:0位
0xf59	KEY3_1	OTP访问密钥3（ECC）的第31:16位
0xf5a	KEY3_2	OTP访问密钥3（ECC）的第47:32位
0xf5b	KEY3_3	OTP访问密钥3（ECC）的第63:48位
0xf5c	KEY3_4	OTP访问密钥3（ECC）的第79:64位
0xf5d	KEY3_5	OTP访问密钥3（ECC）的第95:80位
0xf5e	KEY3_6	OTP访问密钥3（ECC）的第111:96位
0xf5f	KEY3_7	OTP访问密钥3（ECC）的第127:112位
0xf60	KEY4_0	OTP访问密钥4（ECC）的第15:0位
0xf61	KEY4_1	OTP访问密钥4（ECC）的第31至16位
0xf62	KEY4_2	OTP访问密钥4（ECC）的第47至32位
0xf63	KEY4_3	OTP访问密钥4（ECC）的第63至48位
0xf64	KEY4_4	OTP访问密钥4（ECC）的第79至64位
0xf65	KEY4_5	OTP访问密钥4（ECC）的第95至80位
0xf66	KEY4_6	OTP访问密钥4（ECC）的第111至96位
0xf67	KEY4_7	OTP访问密钥4（ECC）的第127至112位
0xf68	KEY5_0	OTP访问密钥5（ECC）的第15至0位
0xf69	KEY5_1	OTP访问密钥5（ECC）的第31至16位
0xf6a	KEY5_2	OTP访问密钥5（ECC）第47至32位
0xf6b	KEY5_3	OTP访问密钥5（ECC）第63至48位
0xf6c	KEY5_4	OTP访问密钥5（ECC）第79至64位
0xf6d	KEY5_5	OTP访问密钥5（ECC）第95至80位
0xf6e	KEY5_6	OTP访问密钥5（ECC）第111至96位
0xf6f	KEY5_7	OTP访问密钥5（ECC）第127至112位
0xf70	KEY6_0	OTP访问密钥6（ECC）第15至0位
0xf71	KEY6_1	OTP访问密钥6（ECC）第31至16位
0xf72	KEY6_2	OTP访问密钥6（ECC）第47至32位
0xf73	KEY6_3	OTP访问密钥6（ECC）第63至48位
0xf74	KEY6_4	OTP访问密钥6（ECC）第79至64位
0xf75	KEY6_5	OTP访问密钥6（ECC）第95至80位
0xf76	KEY6_6	OTP访问密钥6（ECC）第111至96位
0xf77	KEY6_7	OTP访问密钥6（ECC）第127至112位
0xf79	KEY1_VALID	密钥1的有效标志
0xf7a	KEY2_VALID	密钥2的有效标志
0xf7b	KEY3_VALID	密钥3的有效标志
0xf7c	KEY4_VALID	密钥4的有效标志

偏移量	名称	说明
0xf7d	KEY5_VALID	键 5 的有效标志。
0xf7e	KEY6_VALID	键 6 的有效标志。
0xf80	PAGE0_LOCK0	页 0 的锁定配置低字节（行 0x0 至 0x3f）。
0xf81	PAGE0_LOCK1	页 0 的锁定配置高字节（行 0x0 至 0x3f）。
0xf82	PAGE1_LOCK0	页 1 的锁定配置低字节（行 0x40 至 0x7f）。
0xf83	PAGE1_LOCK1	页 1 的锁定配置高字节（行 0x40 至 0x7f）。
0xf84	PAGE2_LOCK0	页 2 的锁定配置低字节（行 0x80 至 0xbf）。
0xf85	PAGE2_LOCK1	页 2 的锁定配置高字节（行 0x80 至 0xbf）。
0xf86	PAGE3_LOCK0	页 3 的锁定配置低字节（行 0xc0 至 0xff）。
0xf87	PAGE3_LOCK1	锁定第 3 页的配置高字节（行 0xc0 至 0xff）。
0xf88	PAGE4_LOCK0	锁定第 4 页的配置低字节（行 0x100 至 0x13f）。
0xf89	PAGE4_LOCK1	锁定第 4 页的配置高字节（行 0x100 至 0x13f）。
0xf8a	PAGE5_LOCK0	锁定第 5 页的配置低字节（行 0x140 至 0x17f）。
0xf8b	PAGE5_LOCK1	锁定第 5 页的配置高字节（行 0x140 至 0x17f）。
0xf8c	PAGE6_LOCK0	锁定第 6 页的配置低字节（行 0x180 至 0x1bf）。
0xf8d	PAGE6_LOCK1	锁定第 6 页的配置高字节（行 0x180 至 0x1bf）。
0xf8e	PAGE7_LOCK0	锁定第 7 页的配置低字节（行 0x1c0 至 0x1ff）。
0xf8f	PAGE7_LOCK1	锁定第7页配置的MSB（行0x1c0至0x1ff）
0xf90	PAGE8_LOCK0	锁定第8页配置的LSB（行0x200至0x23f）
0xf91	PAGE8_LOCK1	锁定第8页配置的MSB（行0x200至0x23f）
0xf92	PAGE9_LOCK0	锁定第9页配置的LSB（行0x240至0x27f）
0xf93	PAGE9_LOCK1	锁定第9页配置的MSB（行0x240至0x27f）
0xf94	PAGE10_LOCK0	锁定第10页配置的LSB（行0x280至0x2bf）
0xf95	PAGE10_LOCK1	锁定第10页配置的MSB（行0x280至0x2bf）
0xf96	PAGE11_LOCK0	锁定第11页配置的LSB（行0x2c0至0x2ff）
0xf97	PAGE11_LOCK1	锁定第11页（行0x2c0至0x2ff）的配置高位。
0xf98	PAGE12_LOCK0	锁定第12页（行0x300至0x33f）的配置低位。
0xf99	PAGE12_LOCK1	锁定第12页（行0x300至0x33f）的配置高位。
0xf9a	PAGE13_LOCK0	锁定第13页（行0x340至0x37f）的配置低位。
0xf9b	PAGE13_LOCK1	锁定第13页（行0x340至0x37f）的配置高位。

偏移量	名称	说明
0xf9c	PAGE14_LOCK0	锁定第14页（行0x380至0x3bf）的配置低位。
0xf9d	PAGE14_LOCK1	锁定第14页（行0x380至0x3bf）的配置高位。
0xf9e	PAGE15_LOCK0	锁定第15页（行0x3c0至0x3ff）的配置低位。
0xf9f	PAGE15_LOCK1	锁定第15页（行0x3c0至0x3ff）的配置高位字节。
0xfa0	PAGE16_LOCK0	锁定第16页（行0x400至0x43f）的配置低位字节。
0xfa1	PAGE16_LOCK1	锁定第16页（行0x400至0x43f）的配置高位字节。
0xfa2	PAGE17_LOCK0	锁定第17页（行0x440至0x47f）的配置低位字节。
0xfa3	PAGE17_LOCK1	锁定第17页（行0x440至0x47f）的配置高位字节。
0xfa4	PAGE18_LOCK0	锁定第18页（行0x480至0x4bf）的配置低位字节。
0xfa5	PAGE18_LOCK1	锁定第18页（行0x480至0x4bf）的配置高位字节。
0xfa6	PAGE19_LOCK0	锁定第19页（行0x4c0至0x4ff）的配置低位字节。
0xfa7	PAGE19_LOCK1	锁定第19页（行0x4c0至0x4ff）配置的高位。
0xfa8	PAGE20_LOCK0	锁定第20页（行0x500至0x53f）配置的低位。
0xfa9	PAGE20_LOCK1	锁定第20页（行0x500至0x53f）配置的高位。
0xfaa	PAGE21_LOCK0	锁定第21页（行0x540至0x57f）配置的低位。
0xfb0	PAGE21_LOCK1	锁定第21页（行0x540至0x57f）配置的高位。
0xfc0	PAGE22_LOCK0	锁定第22页（行0x580至0x5bf）配置的低位。
0xfc1	PAGE22_LOCK1	锁定第22页（行0x580至0x5bf）配置的高位。
0xfae	PAGE23_LOCK0	锁定第23页（行0x5c0至0x5ff）配置的低位。
0xfaf	PAGE23_LOCK1	锁定第23页（行0x5c0至0x5ff）的配置高位字节。
0xfb0	PAGE24_LOCK0	锁定第24页（行0x600至0x63f）的配置低位字节。
0xfb1	PAGE24_LOCK1	锁定第24页（行0x600至0x63f）的配置高位字节。
0xfb2	PAGE25_LOCK0	锁定第25页（行0x640至0x67f）的配置低位字节。

偏移量	名称	说明
0xfb3	PAGE25_LOCK1	锁定第25页（行0x640至0x67f）的配置高位字节。
0xfb4	PAGE26_LOCK0	锁定第26页（行0x680至0x6bf）的配置低位字节。
0xfb5	PAGE26_LOCK1	锁定第26页（行0x680至0x6bf）的配置高位字节。
0xfb6	PAGE27_LOCK0	锁定第27页（行0x6c0至0x6ff）的配置低位字节。
0xfb7	PAGE27_LOCK1	锁定第27页配置的高位（行号 0x6c0 至 0x6ff）。
0xfb8	PAGE28_LOCK0	锁定第28页配置的低位（行号 0x700 至 0x73f）。
0xfb9	PAGE28_LOCK1	锁定第28页配置的高位（行号 0x700 至 0x73f）。
0xfa	PAGE29_LOCK0	锁定第29页配置的低位（行号 0x740 至 0x77f）。
0xfb	PAGE29_LOCK1	锁定第29页配置的高位（行号 0x740 至 0x77f）。
0xfc	PAGE30_LOCK0	锁定第30页配置的低位（行号 0x780 至 0x7bf）。
0xfd	PAGE30_LOCK1	锁定第30页配置的高位（行号 0x780 至 0x7bf）。
0fbe	PAGE31_LOCK0	锁定第31页配置的低位（行号 0x7c0 至 0x7ff）。
0xfb	PAGE31_LOCK1	锁定第31页（行0x7c0至0x7ff）的配置高位（MSB）。
0xfc0	PAGE32_LOCK0	锁定第32页（行0x800至0x83f）的配置低位（LSB）。
0xfc1	PAGE32_LOCK1	锁定第32页（行0x800至0x83f）的配置高位（MSB）。
0xfc2	PAGE33_LOCK0	锁定第33页（行0x840至0x87f）的配置低位（LSB）。
0xfc3	PAGE33_LOCK1	锁定第33页（行0x840至0x87f）的配置高位（MSB）。
0xfc4	PAGE34_LOCK0	锁定第34页（行0x880至0x8bf）的配置低位（LSB）。
0xfc5	PAGE34_LOCK1	锁定第34页（行0x880至0x8bf）的配置高位（MSB）。
0xfc6	PAGE35_LOCK0	锁定第35页（行0x8c0至0x8ff）的配置低位（LSB）。
0xfc7	PAGE35_LOCK1	锁定第35页配置高位（行0x8c0至0x8ff）。
0xfc8	PAGE36_LOCK0	锁定第36页配置低位（行0x900至0x93f）。
0xfc9	PAGE36_LOCK1	锁定第36页配置高位（行0x900至0x93f）。

偏移量	名称	说明
0xfc <sub>a</sub>	PAGE37_LOCK0	锁定第37页配置低位（行0x940至0x97f）。
0xfc <sub>b</sub>	PAGE37_LOCK1	锁定第37页配置高位（行0x940至0x97f）。
0xfc <sub>c</sub>	PAGE38_LOCK0	锁定第38页配置低位（行0x980至0x9bf）。
0xfc <sub>d</sub>	PAGE38_LOCK1	锁定第38页配置高位（行0x980至0x9bf）。
0xfc <sub>e</sub>	PAGE39_LOCK0	锁定第39页配置低位（行0x9c0至0x9ff）。
0xfc <sub>f</sub>	PAGE39_LOCK1	锁定第39页（第0x9c0至0x9ff行）的配置高位（MSB）。
0xfd <sub>0</sub>	PAGE40_LOCK0	锁定第40页（第0xa00至0xa3f行）的配置低位（LSB）。
0xfd <sub>1</sub>	PAGE40_LOCK1	锁定第40页（第0xa00至0xa3f行）的配置高位（MSB）。
0xfd <sub>2</sub>	PAGE41_LOCK0	锁定第41页（第0xa40至0xa7f行）的配置低位（LSB）。
0xfd <sub>3</sub>	PAGE41_LOCK1	锁定第41页（第0xa40至0xa7f行）的配置高位（MSB）。
0xfd <sub>4</sub>	PAGE42_LOCK0	锁定第42页（第0xa80至0xabf行）的配置低位（LSB）。
0xfd <sub>5</sub>	PAGE42_LOCK1	锁定第42页（第0xa80至0xabf行）的配置高位（MSB）。
0xfd <sub>6</sub>	PAGE43_LOCK0	锁定第43页（第0xac0至0xaff行）的配置低位（LSB）。
0xfd <sub>7</sub>	PAGE43_LOCK1	锁定第43页（第0xac0至0xaff行）的配置高位字节（MSB）。
0xfd <sub>8</sub>	PAGE44_LOCK0	锁定第44页（行0xb00至0xb3f）的配置低位字节（LSB）。
0xfd <sub>9</sub>	PAGE44_LOCK1	锁定第44页（行0xb00至0xb3f）的配置高位字节（MSB）。
0xfd <sub>a</sub>	PAGE45_LOCK0	锁定第45页（行0xb40至0xb7f）的配置低位字节（LSB）。
0xfd <sub>b</sub>	PAGE45_LOCK1	锁定第45页（行0xb40至0xb7f）的配置高位字节（MSB）。
0xfd <sub>c</sub>	PAGE46_LOCK0	锁定第46页（行0xb80至0xbbf）的配置低位字节（LSB）。
0xfd <sub>d</sub>	PAGE46_LOCK1	锁定第46页（行0xb80至0xbbf）的配置高位字节（MSB）。
0xfd <sub>e</sub>	PAGE47_LOCK0	锁定第47页（行0xbc0至0xbff）的配置低位字节（LSB）。
0xfd <sub>f</sub>	PAGE47_LOCK1	锁定第47页（行0xbc0至0xbff）的配置高位。
0xfe <sub>0</sub>	PAGE48_LOCK0	锁定第48页（行0xc00至0xc3f）的配置低位。

偏移量	名称	说明
0xfe1	PAGE48_LOCK1	锁定第48页（行0xc00至0xc3f）的配置高位。
0xfe2	PAGE49_LOCK0	锁定第49页（行0xc40至0xc7f）的配置低位。
0xfe3	PAGE49_LOCK1	锁定第49页（行0xc40至0xc7f）的配置高位。
0xfe4	PAGE50_LOCK0	锁定第50页（行0xc80至0xcbf）的配置低位。
0xfe5	PAGE50_LOCK1	锁定第50页（行0xc80至0xcbf）的配置高位。
0xfe6	PAGE51_LOCK0	锁定第51页（行0xcc0至0cff）的配置低位。
0xfe7	PAGE51_LOCK1	锁定第51页（行0xcc0至0cff）的配置高位。
0xfe8	PAGE52_LOCK0	页面52的锁定配置最低有效字节（行0xd00至0xd3f）。
0xfe9	PAGE52_LOCK1	页面52的锁定配置最高有效字节（行0xd00至0xd3f）。
0fea	PAGE53_LOCK0	页面53的锁定配置最低有效字节（行0xd40至0xd7f）。
0feb	PAGE53_LOCK1	页面53的锁定配置最高有效字节（行0xd40至0xd7f）。
0fec	PAGE54_LOCK0	页面54的锁定配置最低有效字节（行0xd80至0xdbf）。
0fed	PAGE54_LOCK1	页面54的锁定配置最高有效字节（行0xd80至0xdbf）。
0fee	PAGE55_LOCK0	页面55的锁定配置最低有效字节（行0xdc0至0xdff）。
0fef	PAGE55_LOCK1	页面55的锁定配置最高有效字节（行0xdc0至0xdff）。
0xff0	PAGE56_LOCK0	页面56的锁定配置最低有效字节（行0xe00至0xe3f）。
0xff1	PAGE56_LOCK1	锁定第56页（行0xe00至0xe3f）的配置高位字节。
0xff2	PAGE57_LOCK0	锁定第57页（行0xe40至0xe7f）的配置低位字节。
0xff3	PAGE57_LOCK1	锁定第57页（行0xe40至0xe7f）的配置高位字节。
0xff4	PAGE58_LOCK0	锁定第58页（行0xe80至0xebf）的配置低位字节。
0xff5	PAGE58_LOCK1	锁定第58页（行0xe80至0xebf）的配置高位字节。
0xff6	PAGE59_LOCK0	锁定第59页（行0xec0至0xeff）的配置低位字节。
0xff7	PAGE59_LOCK1	锁定第59页（行0xec0至0xeff）的配置高位字节。

偏移量	名称	说明
0xff8	PAGE60_LOCK0	锁定第60页（行0xf00至0xf3f）的配置低位字节。
0xff9	PAGE60_LOCK1	锁定第60页（行0xf00至0xf3f）的配置高位字节。
0xffa	PAGE61_LOCK0	锁定第61页（行地址0xf40至0xf7f）配置的低位。
0ffb	PAGE61_LOCK1	锁定第61页（行地址0xf40至0xf7f）配置的高位。
0ffc	PAGE62_LOCK0	锁定第62页（行地址0xf80至0xfb0）配置的低位。
0ffd	PAGE62_LOCK1	锁定第62页（行地址0xf80至0xfb0）配置的高位。
0ffe	PAGE63_LOCK0	锁定第63页（行地址0xfc0至0xffff）配置的低位。
0fff	PAGE63_LOCK1	锁定第63页（行地址0xfc0至0xffff）配置的高位。

## OTP\_DATA：CHIPID0寄存器

偏移：0x000

表1363。CHIPID0  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	公共设备ID的第15至0位。（ECC）  CHIPID0至CHIPID3行包含该芯片的64位随机标识符，可通过USB引导加载程序PICOBLOCK接口或get_sys_info只读存储器API读取。  随机位数使得出现重复ID的可能性极低：例如，在1亿台设备中，99.97%的概率不存在重复ID。该概率被估计低于顺序随机ID分配过程中的工艺错误概率，实际应用中CHIPID可视为唯一标识。	只读	-

## OTP\_DATA：CHIPID1寄存器

偏移：0x001

表1364。CHIPID1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	公共设备ID（ECC）的第31至16位	只读	-

## OTP\_DATA：CHIPID2寄存器

偏移：0x002

表1365. CHIPID2  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	公共设备ID（ECC）的第47至32位	只读	-

## OTP\_DATA：CHIPID3寄存器

偏移: 0x003

表1366. CHIPID3  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	公共设备ID（ECC）的第63至48位	只读	-

## OTP\_DATA：RANDID0寄存器

偏移量: 0x004

表1367. RANDID0  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数（ECC）的第15至0位  RANDID0至7行组成了在设备测试期间生成的128位随机数。  该ID不会通过USB PICOBLOCK的GET_INFO命令或只读存储器get_sys_info() API 暴露。但请注意，USB PICOBLOCK的OTP访问点可读取第0页的全部内容， 因此除非通过BOOT_FLAGS0中的DISABLE_BOOTSEL_USB_PICOBLOCK_IFC标志 禁用USB PICOBLOCK接口，否则该值不具备实际隐私 性。	只读	-

## OTP\_DATA：RANDID1寄存器

偏移: 0x005

表1368. RANDID1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数（ECC）的第31至16位	只读	-

## OTP\_DATA：RANDID2寄存器

偏移: 0x006

表1369. RANDID2  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数的第47至32位（ECC）	只读	-

## OTP\_DATA：RANDID3寄存器

偏移: 0x007

表1370. RANDID3寄存器

位	描述	类型	复位
31:16	保留。	-	-

位	描述	类型	复位
15:0	私有每设备随机数的第63至48位 (ECC)	只读	-

## OTP\_DATA: RANDID4寄存器

偏移: 0x008

表1371. RANDID4寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数的第79至64位 (ECC)	只读	-

## OTP\_DATA: RANDID5寄存器

偏移: 0x009

表1372. RANDID5寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数的第95至80位 (ECC)	只读	-

## OTP\_DATA: RANDID6寄存器

偏移: 0x00a

表1373. RANDID6寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数的第111至96位 (ECC)	只读	-

## OTP\_DATA: RANDID7寄存器

偏移: 0x00b

表1374. RANDID7寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	私有每设备随机数的第127至112位 (ECC)	只读	-

## OTP\_DATA: ROSC\_CALIB寄存器

偏移量: 0x010

表1375. ROSC\_CALIB寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	制造过程中测量的环振荡器频率, 单位为kHz (ECC)  该测量在室温下, 于ROSC配置寄存器处于复位状态时, 于1.1 V电压下进行。 。	只读	-

## OTP\_DATA: LPOSC\_CALIB寄存器

偏移: 0x011

表 1376。  
LPOS\_CALIB 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	制造过程中测量的低功耗振荡器频率，单位：Hz (ECC)  该测量在室温下，于LPOS_C微调寄存器处于复位状态时，于1.1 V电压下进行。	只读	-

## OTP\_DATA：NUM\_GPIOS 寄存器

偏移量: 0x018

表 1377。  
NUM\_GPIOS 寄存器

位	描述	类型	复位
31:8	保留。	-	-
7:0	主用户GPIO数量 (bank 0)。QFN80封装中应为48，QFN60封装中应为30。 (ECC)	只读	-

## OTP\_DATA：INFO\_CRC0 寄存器

偏移: 0x036

表 1378。  
INFO\_CRC0 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 地址 0x00 至 0x6b 的 CRC32 低 16 位 (多项式 0x4c11db7，输入反射，输出反射，初始种子全为 1，最终 XOR 全为 1) (ECC)	只读	-

## OTP\_DATA：INFO\_CRC1 寄存器

偏移: 0x037

表 1379。  
INFO\_CRC1 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 地址 0x00 至 0x6b 的 CRC32 高 16 位 (ECC)	只读	-

## OTP\_DATA：CRITO 寄存器

偏移: 0x038

### 说明

第0页关键启动标志 (RBIT-8)

表 1380。CRITO  
寄存器

位	描述	类型	复位
31:2	保留。	-	-
1	<b>RISCV_DISABLE</b> : 永久禁用 RISC-V 处理器 (Hazard3)	只读	-
0	<b>ARM_DISABLE</b> : 永久禁用 ARM 处理器 (Cortex-M33)	只读	-

## OTP\_DATA：CRITO\_R1、CRITO\_R2、...、CRITO\_R6、CRITO\_R7 寄存器

偏移量: 0x039、0x03a、...、0x03e、0x03f

表 1381。CRIT0\_R1、CRIT0\_R2、...、CRIT0\_R6、CRIT0\_R7 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	CRIT0 的冗余副本	只读	-

## OTP\_DATA：CRIT1 寄存器

偏移量: 0x040

### 说明

第1页关键启动标志 (RBIT-8)

表 1382。CRIT1 寄存器

位	描述	类型	复位
31:7	保留。	-	-
6:5	<b>GLITCH_DETECTOR_SENS</b> : 将故障检测器的灵敏度提高至默认值以上	只读	-
4	<b>GLITCH_DETECTOR_ENABLE</b> : 启用故障检测器，检测到异常时钟或电源事件时复位系统	只读	-
3	<b>BOOT_ARCH</b> : 设置默认启动架构，0=ARM，1=RISC-V 如果设置了 ARM_DISABLE、RISCV_DISABLE 或 SECURE_BOOT_ENABLE，则忽略。	只读	-
2	<b>DEBUG_DISABLE</b> : 禁用所有调试访问	只读	-
1	<b>SECURE_DEBUG_DISABLE</b> : 禁用安全调试访问	只读	-
0	<b>SECURE_BOOT_ENABLE</b> : 启用启动签名校验，并永久禁用 RISC-V 内核。	只读	-

## OTP\_DATA：CRIT1\_R1、CRIT1\_R2、...、CRIT1\_R6、CRIT1\_R7 寄存器

偏移量: 0x041、0x042、...、0x046、0x047

表 1383。CRIT1\_R1、CRIT1\_R2、...、CRIT1\_R6、CRIT1\_R7 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	CRIT1 的冗余副本	只读	-

## OTP\_DATA：BOOT\_FLAGS0 寄存器

偏移量: 0x048

### 说明

禁用/启用 RP2350 只读存储器中的启动路径/功能。禁用状态始终优先于启用。当OTP中存在必须有效的其他配置时，提供启用功能。(RBIT-3)

表 1384。BOOT\_FLAGS0 寄存器

位	描述	类型	复位
31:22	保留。	-	-
21	<b>DISABLE_SRAM_WINDOW_BOOT</b>	只读	-
20	<b>DISABLE_XIP_ACCESS_ON_SRAM_ENTRY</b> : 进入 SRAM 二进制文件后禁用所有对 XIP 的访问  请注意，这将导致访问 XIP 的 bootrom API 失败，包括与分区表交互的 API。 。	只读	-
19	<b>DISABLE_BOOTSEL_UART_BOOT</b>	只读	-

位	描述	类型	复位
18	<b>DISABLE_BOOTSEL_USB_PICOBOOT_IFC</b>	只读	-
17	<b>DISABLE_BOOTSEL_USB_MSD_IFC</b>	只读	-
16	<b>DISABLE_WATCHDOG_SCRATCH</b>	只读	-
15	<b>DISABLE_POWER_SCRATCH</b>	只读	-
14	<p><b>ENABLE OTP BOOT</b>: 启用 OTP 启动。将从 OTPBOOT_SRC 指定的位置开始，加载由 OTPBOOT_LEN 指定数量的 OTP 行到由 OTPBOOT_DST1 和 OTPBOOT_DST0 指定的 SRAM 位置。</p> <p>加载的程序映像以 ECC 形式存储，每行 16 位，且必须包含有效的 IMAG_E_DEF。未经先行在 OTP 中烧录映像并配置 OTPBOOT_LEN、OTPBOOT_SRC、OTPBOOT_DST0 和 OTPBOOT_DST1，严禁设置此位。</p> <p>请注意，OTPBOOT_LEN 和 OTPBOOT_SRC 必须是偶数行 OTP。等效地，映像大小必须为 32 位的整数倍，且必须从 ECC 读取数据地址窗口中 32 位对齐的地址开始。</p>	只读	-
13	<b>DISABLE OTP BOOT</b> : 优先于 ENABLE OTP BOOT。	只读	-
12	<b>DISABLE_FLASH_BOOT</b>	只读	-
11	<b>ROLLBACK_REQUIRED</b> : 要求二进制文件必须包含回滚版本信息。首次启动具有回滚版本的二进制文件时，自动设置此项。	只读	-
10	<b>HASHED_PARTITION_TABLE</b> : 要求对分区表进行哈希处理（若未签名）。	只读	-
9	<b>SECURE_PARTITION_TABLE</b> : 要求对分区表进行签名。	只读	-
8	<b>DISABLE_AUTO_SWITCH_ARCH</b> : 当启动的（唯一）二进制文件属于另一Arm/RISC-V架构且两种架构均已启用时，禁用CPU架构自动切换。	只读	-
7	<b>SINGLE_FLASH_BINARY</b> : 限制闪存启动路径仅使用闪存起始处的单个二进制文件。	只读	-
6	<p><b>OVERRIDE_FLASH_PARTITION_SLOT_SIZE</b>: 覆盖默认的闪存元数据扫描大小限制。</p> <p>该值由FLASH_PARTITION_SLOT_SIZE指定。设置此位前，请确保FLASH_PARTITION_SLOT_SIZE的有效性。</p>	只读	-
5	<b>FLASH_DEVINFO_ENABLE</b> : 标记FLASH_DEVINFO包含有效且经过ECC校验的描述外部闪存设备的数据。	只读	-
4	<b>FAST_SIGCHECK_ROSC_DIV</b> : 启用签名校验期间ROSC除数的四分之一分频，以缩短安全启动时间。	只读	-
3	<p><b>FLASH_IO_VOLTAGE_1V8</b>: 若设为1，首次通过bootrom访问闪存时，将通过VOLTAGE_SELECT寄存器配置QSPI引脚组为1.8V工作电压。此项设置可稍微改善低电压下引脚的输入时序，但不影响其输出特性。</p> <p>若设置为0，则保持VOLTAGE_SELECT处于复位状态（适用于2.5 V及以上电压下的操作）。</p>	只读	-

位	描述	类型	复位
2	<b>ENABLE_BOOTSEL_NON_DEFAULT_PLL_XOSC_CFG</b> : 在进入BOOTSEL模式前启用加载非默认XOSC和PLL配置。  设置此位之前, 请确保BOOTSEL_XOSC_CFG和BOOTSEL_PLL_CFG已正确编程。  若设置此位, 用户软件可使用BOOTSEL_PLL_CFG的内容, 根据固定的48 MHz USB启动频率计算预期的XOSC频率。	只读	-
1	<b>ENABLE_BOOTSEL_LED</b> : 启用引导加载器活动指示灯。若设置, 则假定bootsel_led_cfg有效。	只读	-
0	保留。	-	-

## OTP\_DATA: BOOT\_FLAGS0\_R1、BOOT\_FLAGS0\_R2寄存器

偏移量: 0x049、0x04a

表1385。  
BOOT\_FLAGS0\_R1,  
BOOT\_FLAGS0\_R2  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	BOOT_FLAGS0的冗余副本	只读	-

## OTP\_DATA: BOOT\_FLAGS1 寄存器

偏移量: 0x04b

### 描述

禁用/启用 RP2350 只读存储器中的启动路径/功能。禁用状态始终优先于启用。当OTP中存在必须有效的其他配置时, 提供启用功能。 (RBIT-3)

表 1386。  
BOOT\_FLAGS1  
寄存器

位	描述	类型	复位
31:20	保留。	-	-
19	<b>DOUBLE_TAP</b> : 通过对 RUN/RSTn 引脚进行双击, 启用进入 BOOTS EL 模式。此功能会根据 DOUBLE_TAP_DELAY 的配置, 显著延长启动时间。  该功能通过在启动时 (即复位后) 等待, 检测是否很快施加了第二次复位来实现。第二次复位由只读存储器借助 POWMAN_CHIP_RESET_DOUBLE_TAP 标志检测, 该标志不会被外部复位引脚复位, 且只读存储器进入 BOOTSEL 模式 (NSBOOT), 以通过 USB 或 UART 等待进一步指令。	只读	-
18:16	<b>DOUBLE_TAP_DELAY</b> : 调整启用 DOUBLE_TAP 双击进入 BOOTSEL 模式时, 等待第二次复位的时间。最小值为 50 毫秒, 该字段的每个单位增加 50 毫秒。  例如, 将此字段设置为最大值 7, 会导致芯片启动时等待 400 毫秒, 以检测请求进入 BOOTSEL 模式的第二次复位。  200 milliseconds (DOUBLE_TAP_DELAY=3) 是一个合适的中间值。	只读	-
15:12	保留。	-	-

位	描述	类型	复位
11:8	<p><b>KEY_INVALID:</b> 将启动密钥标记为无效，或防止其成为有效密钥。在安全启动的签名校验过程中，bootrom 将忽略所有被标记为无效的启动密钥。</p> <p>该字段中的每个位对应于存储于 OTP 第2页的四个256位启动密钥哈希之一。</p> <p>在配置启动密钥时，建议将不打算使用的密钥槽标记为 KEY_INVALID，以防止之后安装无关的密钥。</p>	只读	-
7:4	保留。	-	-
3:0	<p><b>KEY_VALID:</b> 将所有可能的启动密钥标记为有效。bootrom 将针对所有有效的启动密钥进行签名校验，忽略无效的启动密钥。</p> <p>该字段中的每个位对应于存储于 OTP 第2页的四个256位启动密钥哈希之一。</p> <p>如果对应的 KEY_INVALID 位被设置，则 KEY_VALID 位将被忽略。启动密钥仅在设置 KEY_VALID 且未设置 KEY_INVALID 时视为有效。</p> <p>若启动密钥不包含您的 secp256k1 公钥的有效 SHA-256 哈希，请勿将其标记为 KEY_VALID。请在设置 KEY_VALID 位前，编程后务必验证密钥——若启动密钥存在不可纠正的 ECC 错误且启用了安全启动，设备将无法启动。</p> <p>未预装有效密钥前，请勿启用安全启动。否则设备将无法启动。</p>	只读	-

## OTP\_DATA: BOOT\_FLAGS1\_R1, BOOT\_FLAGS1\_R2 寄存器

偏移量: 0x04c, 0x04d

表 1387。  
BOOT\_FLAGS1\_R1,  
BOOT\_FLAGS1\_R2  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	BOOT_FLAGS1 的冗余副本	只读	-

## OTP\_DATA: DEFAULT\_BOOT\_VERSION0 寄存器

偏移量: 0x04e

表 1388。  
DEFAULT\_BOOT\_VERS  
ION0 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	默认启动版本温度计计数器，位23:0 (RBIT-3)	只读	-

## OTP\_DATA: DEFAULT\_BOOT\_VERSION0\_R1, DEFAULT\_BOOT\_VERSION0\_R2 寄存器

偏移量: 0x04f, 0x050

表 1389。  
`DEFAULT_BOOT_VERS`  
`ION0_R1,`  
`DEFAULT_BOOT_VERS`  
`ION0_R2 寄存器`

位	描述	类型	复位
31:24	保留。	-	-
23:0	DEFAULT_BOOT_VERSION0的冗余副本	只读	-

## OTP\_DATA: DEFAULT\_BOOT\_VERSION1 寄存器

偏移量: 0x051

表 1390。  
`DEFAULT_BOOT_VERS`  
`ION1 寄存器`

位	描述	类型	复位
31:24	保留。	-	-
23:0	默认启动版本温度计计数器, 位47:24 (RBIT-3)	只读	-

## OTP\_DATA: DEFAULT\_BOOT\_VERSION1\_R1, DEFAULT\_BOOT\_VERSION1\_R2 寄存器

偏移量: 0x052, 0x053

表 1391。  
`DEFAULT_BOOT_VERS`  
`ION1_R1,`  
`DEFAULT_BOOT_VERS`  
`ION1_R2 寄存器`

位	描述	类型	复位
31:24	保留。	-	-
23:0	DEFAULT_BOOT_VERSION1的冗余副本	只读	-

## OTP\_DATA: FLASH\_DEVINFO 寄存器

偏移: 0x054

### 描述

存储有关外部闪存设备的信息。 (ECC) 如果设置了BO

OT\_FLAGS0\_FLASH\_DEVINFO\_ENABLE, 则假定其有效。

表 1392。  
`FLASH_DEVINFO`  
 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:12	<b>CS1_SIZE:</b> 芯片选择1上的闪存/PSRAM设备大小 (地址范围为0x11000000至0x11fffff)。  值为零时解码为大小为零 (无设备)。 非零值被解码为4kiB << CS1_SIZE。 例如, 四兆字节的设备用CS1_SIZE值10编码, 十六兆字节的设备用CS1_SIZE值12编码。  当 BOOT_FLAGS0_FLASH_DEVINFO_ENABLE 未设置时, 默认为零。	只读	-
	枚举值:		
	0x0 → 无		
	0x1 → 8K		
	0x2 → 16K		
	0x3 → 32K		
	0x4 → 64K		
	0x5 → 128K		

位	描述	类型	复位
	0x6 → 256K		
	0x7 → 512K		
	0x8 → 1M		
	0x9 → 2M		
	0xa → 4M		
	0xb → 8M		
	0xc → 16M		
11:8	<p><b>CS0_SIZE:</b> 芯片选择0上的闪存/PSRAM设备大小（地址范围为0x10000000至0x10fffff）。</p> <p>值为零时解码为大小为零（无设备）。非零值按4kiB &lt;&lt; CS0_SIZE 解码。例如，四兆字节对应的 CS0_SIZE 值为10，十六兆字节对应的 CS0_SIZE 值为12。</p> <p>当未设置 BOOT_FLAGS0_FLASH_DEVINFO_ENABLE 时，默认值为 12（16 MiB）。</p>	只读	-
	枚举值：		
	0x0 → 无		
	0x1 → 8K		
	0x2 → 16K		
	0x3 → 32K		
	0x4 → 64K		
	0x5 → 128K		
	0x6 → 256K		
	0x7 → 512K		
	0x8 → 1M		
	0x9 → 2M		
	0xa → 4M		
	0xb → 8M		
	0xc → 16M		
7	<p><b>D8H_ERASE_SUPPORTED:</b> 如果为真，则假定所有连接设备均支持（或者在 PSRAM 情况下忽略）带有命令前缀 D8h、擦除大小为 64 kiB 且具有 24 位地址的块擦除命令。几乎所有 25 系列闪存设备均支持此命令。</p> <p>如果设置，bootrom 会在可行时使用 D8h 擦除命令以加速批量擦除操作。这将加快闪存编程速度。</p> <p>当未设置 BOOT_FLAGS0_FLASH_DEVINFO_ENABLE 时，该字段默认为假。</p>	只读	-
6	保留。	-	-

位	描述	类型	复位
5:0	<p><b>CS1_GPIO:</b> 指定用于次级闪存片选 (CS1) 的 GPIO 编号，该片选用于选择映射在系统地址 0x11000000 至 0x11fffff 的外部 QSPI 设备。CS0 无此配置，因为主片选拥有专用引脚。</p> <p>在 RP2350 上，允许的 GPIO 编号为 0、8、19 和 47。</p> <p>如果 CS1_size 为零，则忽略。若 CS1_SIZE 非零，启动只读存储器将在进入闪存启动路径或任何可能使用 QSPI 闪存接口的路径（例如 BOOTS EL 模式 (nsboot)）时，自动将该 GPIO 配置为第二片选信号。</p>	只读	-

## OTP\_DATA: FLASH\_PARTITION\_SLOT\_SIZE 寄存器

偏移: 0x055

表 1393。  
FLASH\_PARTITION\_SLOT\_SIZE 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	闪存起始处分区表槽 0 与槽 1 之间的间隙（默认大小为 4096 字节）（ECC）由 BOOT_FLAGS 中的 OVERRIDE_FLASH_PARTITION_SLOT_SIZE 位启用，大小为 $4096 \times (\text{数值} + 1)$	只读	-

## OTP\_DATA: BOOTSEL\_LED\_CFG 寄存器

偏移: 0x056

### 描述

LED 状态的引脚配置，供 USB 引导加载程序使用。（ECC）

如果设置了 BOOT\_FLAGS0\_ENABLE\_BOOTSEL\_LED，则该配置必须有效。

表 1394。  
BOOTSEL\_LED\_CFG 寄存器

位	描述	类型	复位
31:9	保留。	-	-
8	<b>ACTIVELOW:</b> LED 为低电平有效。（默认值：高电平有效）	只读	-
7:6	保留。	-	-
5:0	<b>PIN:</b> 用于引导程序活动 LED 的 GPIO 索引。	只读	-

## OTP\_DATA: BOOTSEL\_PLL\_CFG 寄存器

偏移量: 0x057

### 说明

BOOTSEL 模式的可选 PLL 配置。（ECC）

应根据晶体振荡器频率配置此项，以产生精确的 48 MHz。用户模式软件也可使用该数值，基于假定的 48 MHz PLL 输出计算预期的晶体频率。

如果未提供配置，则默认晶体频率为 12 MHz。

PLL 频率可按以下公式计算：

$$\text{PLL out} = (\text{XOSC frequency} / (\text{REFDIV}+1)) \times \text{FBDIV} / (\text{POSTDIV1} \times \text{POSTDIV2})$$

反之，晶体频率可按以下公式计算：

$$\text{XOSC frequency} = 48 \text{ MHz} \times (\text{REFDIV}+1) \times (\text{POSTDIV1} \times \text{POSTDIV2}) / \text{FBDIV}$$

(注：REFDIV加1是因该OTP位置存储的值为实际除数减一。)

仅在BOOT\_FLAGS0中设置ENABLE\_BOOTSEL\_NON\_DEFAULT\_PLL\_XOSC\_CFG时使用。仅当本行及BOOTSEL\_XOSC\_CFG均正确编程后，方可设置该位。

表 1395。  
BOOTSEL\_PLL\_CFG  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15	<b>REFDIV</b> : PLL参考除数值减一。  编程值为0表示参考除数为1。编程值为1表示参考除数为2（适用于极高速XIN输入）	只读	-
14:12	<b>POSTDIV2</b> : PLL后分频2除数，范围为1至7（含）。	只读	-
11:9	<b>POSTDIV1</b> : PLL后分频1除数，范围为1至7（含）。	只读	-
8:0	<b>FBDIV</b> : PLL反馈除数，范围为16至320（含）。	只读	-

## OTP\_DATA: BOOTSEL\_XOSC\_CFG 寄存器

偏移: 0x058

### 描述

USB引导加载程序的非默认晶体振荡器配置。（ECC）

这些数值亦可由配置晶振的用户代码使用。

仅当BOOT\_FLAGS0中设置了ENABLE\_BOOTSEL\_NON\_DEFAULT\_PLL\_XOSC\_CFG时使用。该位应仅在此项与BOOTSEL\_PLL\_CFG均正确配置后才设置。

表 1396。  
BOOTSEL\_XOSC\_CFG  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:14	范围: XOSC_CTRL_FREQ_RANGE 寄存器的值。	只读	-
	枚举值:		
	0x0 → 1_15MHZ		
	0x1 → 10_30MHZ		
	0x2 → 25_60MHZ		
	0x3 → 40_100MHZ		
13:0	启动: XOSC_STARTUP 寄存器的值。	只读	-

## OTP\_DATA: USB\_BOOT\_FLAGS 寄存器

偏移: 0x059

### 描述

USB引导特定功能标志（RBIT-3）

表 1397。  
USB\_BOOT\_FLAGS  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23	<b>DP_DM_SWAP</b> : USB启动期间交换DM/DP，用于支持具有镜像USB布线（无论是有意还是无意）的板级布局。	只读	-

位	描述	类型	复位
22	<b>WHITE_LABEL_ADDR_VALID</b> : USB_WHITE_LABEL 结构内 INFO_UF2_TXT_BOARD_ID_STRDEF 条目（索引 15）的有效标志	只读	-
21:16	保留。	-	-
15	<b>WL_INFO_UF2_TXT_BOARD_ID_STRDEF_VALID</b> : USB_WHITE_LABEL_ADDR 字段的有效标志	只读	-
14	<b>WL_INFO_UF2_TXT_MODEL_STRDEF_VALID</b> : USB_WHITE_LABEL 结构内 INFO_UF2_TXT_MODEL_STRDEF 条目（索引 14）的有效标志	只读	-
13	<b>WL_INDEX_HTM_REDIRECT_NAME_STRDEF_VALID</b> : USB_WHITE_LABEL 结构内 INDEX_HTM_REDIRECT_NAME_STRDEF 条目（索引 13）的有效标志	只读	-
12	<b>WL_INDEX_HTM_REDIRECT_URL_STRDEF_VALID</b> : USB_WHITE_LABEL 结构中 INDEX_HTM_REDIRECT_URL_STRDEF 条目的有效标志（索引 12）	只读	-
11	<b>WL_SCSI_INQUIRY_VERSION_STRDEF_VALID</b> : USB_WHITE_LABEL 结构中 SCSI_INQUIRY_VERSION_STRDEF 条目的有效标志（索引 11）	只读	-
10	<b>WL_SCSI_INQUIRY_PRODUCT_STRDEF_VALID</b> : USB_WHITE_LABEL 结构中 SCSI_INQUIRY_PRODUCT_STRDEF 条目的有效标志（索引 10）	只读	-
9	<b>WL_SCSI_INQUIRY_VENDOR_STRDEF_VALID</b> : USB_WHITE_LABEL 结构中 SCSI_INQUIRY_VENDOR_STRDEF 条目的有效标志（索引 9）	只读	-
8	<b>WL_VOLUME_LABEL_STRDEF_VALID</b> : USB_WHITE_LABEL 结构中 VOLUME_LABEL_STRDEF 条目的有效标志（索引 8）	只读	-
7	<b>WL_USB_CONFIG_ATTRIBUTES_MAX_POWER_VALUES_VALID</b> : USB_WHITE_LABEL 结构中 USB_CONFIG_ATTRIBUTES_MAX_POWER_VALUES 条目的有效标志（索引 7）	只读	-
6	<b>WL_USB_DEVICE_SERIAL_NUMBER_STRDEF_VALID</b> : USB_WHITE_LABEL 结构中 USB_DEVICE_SERIAL_NUMBER_STRDEF 条目的有效标志（索引 6）	只读	-
5	<b>WL_USB_DEVICE_PRODUCT_STRDEF_VALID</b> : USB_WHITE_LABEL 结构体中 USB_DEVICE_PRODUCT_STRDEF 条目的有效标志 USB_DEVICE_PRODUCT_STRDEF 条目（索引 5）	只读	-
4	<b>WL_USB_DEVICE_MANUFACTURER_STRDEF_VALID</b> : USB_WHITE_LABEL 结构体中 USB_DEVICE_MANUFACTURER_STRDEF 条目的有效标志（索引 4）	只读	-
3	<b>WL_USB_DEVICE_LANG_ID_VALUE_VALID</b> : USB_WHITE_LABEL 结构体中 USB_DEVICE_LANG_ID_VALUE 条目的有效标志 USB_DEVICE_LANG_ID_VALUE 条目（索引 3）	只读	-
2	<b>WL_USB_DEVICE_SERIAL_NUMBER_VALUE_VALID</b> : USB_WHITE_LABEL 结构体中 USB_DEVICE_BCD_DEVICEVALUE 条目的有效标志（索引 2）	只读	-
1	<b>WL_USB_DEVICE_PID_VALUE_VALID</b> : USB_WHITE_LABEL 结构体中 USB_DEVICE_PID_VALUE 条目的有效标志（索引 1）	只读	-

位	描述	类型	复位
0	WL_USB_DEVICE_VID_VALUE_VALID：USB_WHITE_LABEL结构体中USB_DEVICE_VID_VALUE条目的有效标志（索引0）	只读	-

## OTP\_DATA: USB\_BOOT\_FLAGS\_R1, USB\_BOOT\_FLAGS\_R2寄存器

偏移量：0x05a, 0x05b

表1398。  
USB\_BOOT\_FLAGS\_R1,  
USB\_BOOT\_FLAGS\_R2  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:0	USB_BOOT_FLAGS的冗余副本	只读	-

## OTP\_DATA: USB\_WHITE\_LABEL\_ADDR寄存器

偏移: 0x05c

表1399。  
USB\_WHITE\_LABEL\_A  
DDR寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p>OTP中USB_WHITE_LABEL结构的行索引 (ECC)</p> <p>该表包含16行，每行均带ECC，并由USB_BOOT_FLAGS (ECC) 中的对应有效位标注为有效。</p> <p>条目要么为_VALUE类型，其16位值按原样使用，要么为_STRDEF类型，作为指向字符串值的指针。</p> <p>_STRDEF中存储的值由两个独立字节组成：第一个 (LSB) 字节的低七位表示字符串中的字符数，最高位若设置，表明字符串中的每个字符为两个字节 (Unicode)；未设置则为一个字节。第二个 (MSB) 字节表示字符串数据的位置，编码为相对于USB_WHITE_LABEL_ADDR的行数；即字符串起始行是USB_WHITE_LABEL_ADDR值加上该MSB字节。</p> <p>在每种情况下，相应的有效位允许替换由只读存储器提供的对应项的默认值。</p> <p>请注意，Unicode _STRDEF 仅支持USB_DEVICE_PRODUCT_STRDEF、USB_DEVICE_SERIAL_NUMBER_STRDEF 和 USB_DEVICE_MANUFACTURER_STRDEF。如为其他字段指定Unicode值，将被忽略；而这三项的非Unicode值则通过将高8位设为零转换为Unicode字符。</p> <p>请注意，若USB_WHITE_LABEL结构或其对应字符串因基于OTP权限的BOOTS EL模式而不可读取，或未满足对齐要求，则将采用对应的默认值。</p> <p>索引值指示每个字段的位置 (USB_WHITE_LABEL_ADDR 行值 + 索引)：</p> <ul style="list-style-type: none"> <li>枚举值：</li> <li>0x0000 → INDEX_USB_DEVICE_VID_VALUE</li> <li>0x0001 → INDEX_USB_DEVICE_PID_VALUE</li> </ul>	只读	-

位	描述	类型	复位
	0x0002 → INDEX_USB_DEVICE_BCD_DEVICE_VALUE		
	0x0003 → INDEX_USB_DEVICE_LANG_ID_VALUE		
	0x0004 → INDEX_USB_DEVICE_MANUFACTURER_STRDEF		
	0x0005 → INDEX_USB_DEVICE_PRODUCT_STRDEF		
	0x0006 → INDEX_USB_DEVICE_SERIAL_NUMBER_STRDEF		
	0x0007 → INDEX_USB_CONFIG_ATTRIBUTES_MAX_POWER_VALUES		
	0x0008 → INDEX_VOLUME_LABEL_STRDEF		
	0x0009 → INDEX_SCSI_INQUIRY_VENDOR_STRDEF		
	0x000a → INDEX_SCSI_INQUIRY_PRODUCT_STRDEF		
	0x000b → INDEX_SCSI_INQUIRY_VERSION_STRDEF		
	0x000c → INDEX_INDEX_HTM_REDIRECT_URL_STRDEF		
	0x000d → INDEX_INDEX_HTM_REDIRECT_NAME_STRDEF		
	0x000e → INDEX_INFO_UF2_TXT_MODEL_STRDEF		
	0x000f → INDEX_INFO_UF2_TXT_BOARD_ID_STRDEF		

## OTP\_DATA: OTPBOOT\_SRC寄存器

偏移: 0x05e

表1400。  
OTPBOOT\_SRC  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p>OTP引导映像的OTP起始行。 (ECC)</p> <p>如果启用了OTP启动，bootrom将从该位置加载至SRAM，然后直接执行加载的镜像。请注意，若设置了SECURE_BOOT_ENABLE，镜像必须经过签名。镜像本身假定已受ECC保护。</p> <p>该数值必须为偶数。同理，OTP启动镜像必须从ECC读取数据地址窗口中的字对齐位置起始。</p>	只读	-

## OTP\_DATA: OTPBOOT\_LEN寄存器

偏移: 0x05f

表1401。  
OTPBOOT\_LEN  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	<p>OTP启动映像的行长度。 (ECC)</p> <p>OTPBOOT_LEN必须为偶数。镜像总大小必须为4字节（32位）的整数倍。</p>	只读	-

## OTP\_DATA: OTPBOOT\_DST0寄存器

偏移量: 0x060

表1402。  
OTPBOOT\_DST0  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 启动映像加载地址（及入口点）位 15:0。 (ECC)  该位置必须位于主SRAM内（主SRAM地址范围为0x20000000 至0x20082000），且必须字对齐。	只读	-

## OTP\_DATA: OTPBOOT\_DST1 寄存器

偏移地址: 0x061

表 1403。  
OTPBOOT\_DST1  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 启动映像加载地址（及入口点）位 31:16。 (ECC)  该位置必须位于主SRAM内（主SRAM地址范围为0x20000000 至0x20082000），且必须字对齐。	只读	-

## OTP\_DATA: BOOTKEY0\_0, BOOTKEY0\_1, ..., BOOTKEY3\_14, BOOTKEY3\_15 寄存器

偏移地址: 0x080, 0x081, ..., 0x0BE, 0x0BF

表 1404。  
BOOTKEY0\_0,  
BOOTKEY0\_1, ...,  
BOOTKEY3\_14,  
BOOTKEY3\_15  
寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	位 N + 15 : N引导密钥的 SHA-256 哈希值K (ECC)	只读	-

## OTP\_DATA: KEY1\_0, KEY2\_0, ..., KEY5\_0, KEY6\_0 寄存器

偏移地址: 0xF48, 0xF50, ..., 0xF68, 0xF70

表 1405。KEY1\_0, KE  
Y2\_0, ..., KEY5\_0  
, KEY6\_0 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP访问密钥的位 15:0 n ( ECC)	只读	-

## OTP\_DATA: KEY1\_1、KEY2\_1、...、KEY5\_1、KEY6\_1 寄存器

偏移量: 0xf49、0xf51、...、0xf69、0xf71

表 1406 KEY1\_1、KEY  
2\_1、...、KEY5\_1  
, KEY6\_1 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP访问密钥的位 31:16 n ( ECC)	只读	-

## OTP\_DATA: KEY1\_2、KEY2\_2、...、KEY5\_2、KEY6\_2 寄存器

偏移量: 0xf4a、0xf52、...、0xf6a、0xf72

表 1407 KEY1\_2、KEY2\_2、...、KEY5\_2  
、KEY6\_2 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 访问密钥的位 47:32 $n$ (ECC)	只读	-

**OTP\_DATA：KEY1\_3、KEY2\_3、...、KEY5\_3、KEY6\_3 寄存器**

偏移量: 0xf4b、0xf53、...、0xf6b、0xf73

表 1408。KEY1\_3、KEY2\_3、...、KEY5\_3  
、KEY6\_3 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 访问密钥的第 63 至 48 位 $n$ (ECC)	只读	-

**OTP\_DATA：KEY1\_4、KEY2\_4、...、KEY5\_4、KEY6\_4 寄存器**

偏移量: 0xf4c、0xf54、...、0xf6c、0xf74

表 1409。KEY1\_4、KEY2\_4、...、KEY5\_4  
、KEY6\_4 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 访问密钥的第 79 至 64 位 $n$ (ECC)	只读	-

**OTP\_DATA：KEY1\_5、KEY2\_5、...、KEY5\_5、KEY6\_5 寄存器**

偏移量: 0xf4d, 0xf55, ..., 0xf6d, 0xf75

表 1410。KEY1\_5、KEY2\_5、...、KEY5\_5  
、KEY6\_5 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 访问密钥的位 95:80 $n$ (ECC)	只读	-

**OTP\_DATA：KEY1\_6、KEY2\_6、...、KEY5\_6、KEY6\_6 寄存器**

偏移量: 0xf4e, 0xf56, ..., 0xf6e, 0xf76

表 1411。KEY1\_6、KEY2\_6、...、KEY5\_6  
、KEY6\_6 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 访问密钥的位 111:96 $n$ (ECC)	只读	-

**OTP\_DATA：KEY1\_7、KEY2\_7、...、KEY5\_7、KEY6\_7 寄存器**

偏移量: 0xf4f, 0xf57, ..., 0xf6f, 0xf77

表 1412。KEY1\_7、KEY2\_7、...、KEY5\_7  
、KEY6\_7 寄存器

位	描述	类型	复位
31:16	保留。	-	-
15:0	OTP 访问密钥第 127:112 位 $n$ (ECC)	只读	-

**OTP\_DATA：KEY1\_VALID 寄存器**

偏移: 0xf79

**描述**

密钥 1 的有效标志。一旦设定有效标志，该密钥将无法被读取或写入，且成为用于保护 OTP 页的有效固定密钥。

表 1413。  
KEY1\_VALID 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>VALID_R2</b> : VALID 的冗余副本，采用三路多数投票	只读	-
15:9	保留。	-	-
8	<b>VALID_R1</b> : VALID 的冗余副本，采用三路多数投票	只读	-
7:1	保留。	-	-
0	<b>VALID</b>	只读	-

## OTP\_DATA: KEY2\_VALID 寄存器

偏移: 0xf7a

### 描述

密钥 2 的有效标志。一旦设定有效标志，该密钥将无法被读取或写入，且成为用于保护 OTP 页的有效固定密钥。

表 1414。  
KEY2\_VALID 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>VALID_R2</b> : VALID 的冗余副本，采用三路多数投票	只读	-
15:9	保留。	-	-
8	<b>VALID_R1</b> : VALID 的冗余副本，采用三路多数投票	只读	-
7:1	保留。	-	-
0	<b>VALID</b>	只读	-

## OTP\_DATA: KEY3\_VALID 寄存器

偏移量: 0xf7b

### 描述

密钥3的有效标志。一旦设定有效标志，该密钥将无法被读取或写入，且成为用于保护 OTP 页的有效固定密钥。

表 1415。  
KEY3\_VALID 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>VALID_R2</b> : VALID 的冗余副本，采用三路多数投票	只读	-
15:9	保留。	-	-
8	<b>VALID_R1</b> : VALID 的冗余副本，采用三路多数投票	只读	-
7:1	保留。	-	-
0	<b>VALID</b>	只读	-

## OTP\_DATA: KEY4\_VALID 寄存器

偏移量: 0xf7c

### 描述

密钥4的有效标志。一旦设定有效标志，该密钥将无法被读取或写入，且成为用于保护 OTP 页的有效固定密钥。

表1416。  
KEY4\_VALID 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>VALID_R2</b> : VALID 的冗余副本，采用三路多数投票	只读	-
15:9	保留。	-	-
8	<b>VALID_R1</b> : VALID 的冗余副本，采用三路多数投票	只读	-
7:1	保留。	-	-
0	<b>VALID</b>	只读	-

## OTP\_DATA: KEY5\_VALID 寄存器

偏移量: 0xf7d

### 描述

密钥5的有效标志。一旦设定有效标志，该密钥将无法被读取或写入，且成为用于保护 OTP 页的有效固定密钥。

表1417。  
KEY5\_VALID 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>VALID_R2</b> : VALID 的冗余副本，采用三路多数投票	只读	-
15:9	保留。	-	-
8	<b>VALID_R1</b> : VALID 的冗余副本，采用三路多数投票	只读	-
7:1	保留。	-	-
0	<b>VALID</b>	只读	-

## OTP\_DATA: KEY6\_VALID 寄存器

偏移量: 0xf7e

### 描述

密钥6的有效标志。一旦设定有效标志，该密钥将无法被读取或写入，且成为用于保护 OTP 页的有效固定密钥。

表1418。  
KEY6\_VALID 寄存器

位	描述	类型	复位
31:17	保留。	-	-
16	<b>VALID_R2</b> : VALID 的冗余副本，采用三路多数投票	只读	-
15:9	保留。	-	-
8	<b>VALID_R1</b> : VALID 的冗余副本，采用三路多数投票	只读	-
7:1	保留。	-	-
0	<b>VALID</b>	只读	-

## OTP\_DATA: PAGE0\_LOCK0, PAGE1\_LOCK0, ..., PAGE61\_LOCK0, PAGE62\_LOCK0 寄存器

偏移: 0xf80, 0xf82, ..., 0xffa, 0ffc

### 描述

页面 N 的锁定配置最低有效位（行  $0x40 * N$  至  $0x40 * N + 0x3f$ ）。锁定以三路多数表决编码方式存储，因此各位可独立设置。

该 OTP 位置始终可读，且通过自身权限受到写入保护。

表 1419。  
PAGE0\_LOCK0  
, PAGE1\_LOCK0  
, ..., PAGE61\_L  
OCK0, PAGE62  
\_LOCK0 寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>R2</b> : 位 7:0 的冗余副本	只读	-
15:8	<b>R1</b> : 位 7:0 的冗余副本	只读	-
7	保留。	-	-
6	<b>NO_KEY_STATE</b> : 当至少一个密钥已登记至此页面且未输入匹配密钥时的状态 。	只读	-
	枚举值：		
	0x0 → 只读		
	0x1 → 不可访问		
5:3	<b>KEY_R</b> : 必须输入以授予读取权限的硬件密钥索引1至6，若无此类密钥 则为0。	只读	-
2:0	<b>KEY_W</b> : 必须输入以授予写入权限的硬件密钥索引1至6，若无此类密钥 则为0。	只读	-

## OTP\_DATA: PAGE0\_LOCK1, PAGE1\_LOCK1, ..., PAGE61\_LOCK1, PAGE62\_LOCK1 寄存器

偏移量: 0xf81, 0xf83, ..., 0xffb, 0ffd

### 描述

页 N 的锁定配置高位 (行  $0x40 * N$  至  $0x40 * N + 0x3f$ )。锁定以三路多数表决编码方式存储，因此各位可独立设置。

该 OTP 位置始终可读，且通过自身权限受到写入保护。

表 1420。  
PAGE0\_LOCK1,  
PAGE1\_LOCK1, ...  
PAGE61\_LOCK1,  
PAGE62\_LOCK1  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>R2</b> : 位 7:0 的冗余副本	只读	-
15:8	<b>R1</b> : 位 7:0 的冗余副本	只读	-
7:6	保留。	-	-
5:4	<b>LOCK_BL</b> : 为引导程序（包括RP2350 USB引导程序）预留的虚拟锁定位， 用于存储其自身的OTP访问权限。无硬件影响，且无对应的SW_LOCKx寄 存器。	只读	-
	枚举值：		
	0x0 → READ_WRITE: 引导程序允许用户对该页进行读写。		
	0x1 → READ_ONLY: 引导程序允许用户读取该页。		
	0x2 → RESERVED: 禁止使用。行为与INACCESSIBLE相同。		
	0x3 → INACCESSIBLE: 引导程序不允许用户访问该页。		

位	描述	类型	复位
3:2	<p><b>LOCK_NS</b>: 该页非安全访问的锁定状态。采用温度编码，因此锁定状态可通过编程OTP永久从任一状态升级至任何更严格的状态。软件也可通过SW_LOCKx寄存器临时升级锁定状态（直至下一次OTP复位）。</p> <p>注意，READ_WRITE与READ_ONLY在硬件层面等效，因SBPI编程接口不可被非安全软件访问。 然而，安全软件可能会检查这些位，以对非安全OTP编程API施加写入权限。</p>	只读	-
	枚举值：		
	0x0 → READ_WRITE：页面可被非安全软件读取，且安全软件可能允许非安全写入。		
	0x1 → READ_ONLY：页面可被非安全软件读取。		
	0x2 → 保留：禁止使用，行为等同于INACCESSIBLE。		
	0x3 → INACCESSIBLE：页面不可被非安全软件访问。		
1:0	<p><b>LOCK_S</b>: 该页面的安全访问锁定状态。采用温度计编码，锁定状态可通过编程OTP永久升级，从任一状态变更为权限更低的状态。软件亦可通过SW_LOCKx寄存器临时提升锁定状态（直到下一次OTP复位）。</p>	只读	-
	枚举值：		
	0x0 → READ_WRITE：页面可被安全软件完全访问。		
	0x1 → READ_ONLY：页面可被安全软件读取，但不可写入。		
	0x2 → 保留：禁止使用，行为等同于INACCESSIBLE。		
	0x3 → 不可访问：安全软件无法访问该页面。		

## OTP\_DATA: PAGE63\_LOCK0 寄存器

偏移量: 0ffe

### 描述

页面63的锁定配置最低有效位（覆盖行0xfc0至0xfff）。锁定以三路多数表决编码方式存储，因此各位可独立设置。

该OTP位置始终可读，且通过自身权限受到写入保护。

表 1421。  
PAGE63\_LOCK0  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>R2</b> : 位 7:0 的冗余副本	只读	-
15:8	<b>R1</b> : 位 7:0 的冗余副本	只读	-
7	<b>RMA</b> : 对疑似故障设备执行退货维修处理。此操作重新启用工厂测试JTAG接口，并使OTP第3页至第61页永久不可访问。	只读	-
6	<b>NO_KEY_STATE</b> : 当至少一个密钥已登记至此页面且未输入匹配密钥时的状态。	只读	-
	枚举值：		

位	描述	类型	复位
	0x0 → 只读		
	0x1 → 不可访问		
5:3	<b>KEY_R</b> : 必须输入以授予读取权限的硬件密钥索引1至6，若无此类密钥则为0。	只读	-
2:0	<b>KEY_W</b> : 必须输入以授予写入权限的硬件密钥索引1至6，若无此类密钥则为0。	只读	-

## OTP\_DATA: PAGE63\_LOCK1 寄存器

偏移量: 0xffff

### 说明

页面63的锁定配置最高有效位（覆盖行0xfc0至0xffff）。锁定以三路多数表决编码方式存储，因此各位可独立设置。

该 OTP 位置始终可读，且通过自身权限受到写入保护。

表 1422。  
PAGE63\_LOCK1  
寄存器

位	描述	类型	复位
31:24	保留。	-	-
23:16	<b>R2</b> : 位 7:0 的冗余副本	只读	-
15:8	<b>R1</b> : 位 7:0 的冗余副本	只读	-
7:6	保留。	-	-
5:4	<b>LOCK_BL</b> : 为引导程序（包括RP2350 USB引导程序）预留的虚拟锁定位，用于存储其自身的OTP访问权限。无硬件影响，且无对应的SW_LOCKx寄存器。	只读	-
	枚举值：		
	0x0 → READ_WRITE：引导程序允许用户对该页进行读写。		
	0x1 → READ_ONLY：引导程序允许用户读取该页。		
	0x2 → RESERVED：禁止使用。行为与INACCESSIBLE相同。		
	0x3 → INACCESSIBLE：引导程序不允许用户访问该页。		
3:2	<b>LOCK_NS</b> : 该页非安全访问的锁定状态。采用温度编码，因此锁定状态可通过编程OTP永久从任一状态升级至任何更严格的状态。软件也可通过SW_LOCKx寄存器临时升级锁定状态（直至下一次OTP复位）。	只读	-
	注意，READ_WRITE与READ_ONLY在硬件层面等效，因SBPI编程接口不可被非安全软件访问。 然而，安全软件可能会检查这些位，以对非安全OTP编程API施加写入权限。		
	枚举值：		
	0x0 → READ_WRITE：页面可被非安全软件读取，且安全软件可能允许非安全写入。		
	0x1 → READ_ONLY：页面可被非安全软件读取。		
	0x2 → 保留：禁止使用，行为等同于INACCESSIBLE。		
	0x3 → INACCESSIBLE：页面不可被非安全软件访问。		

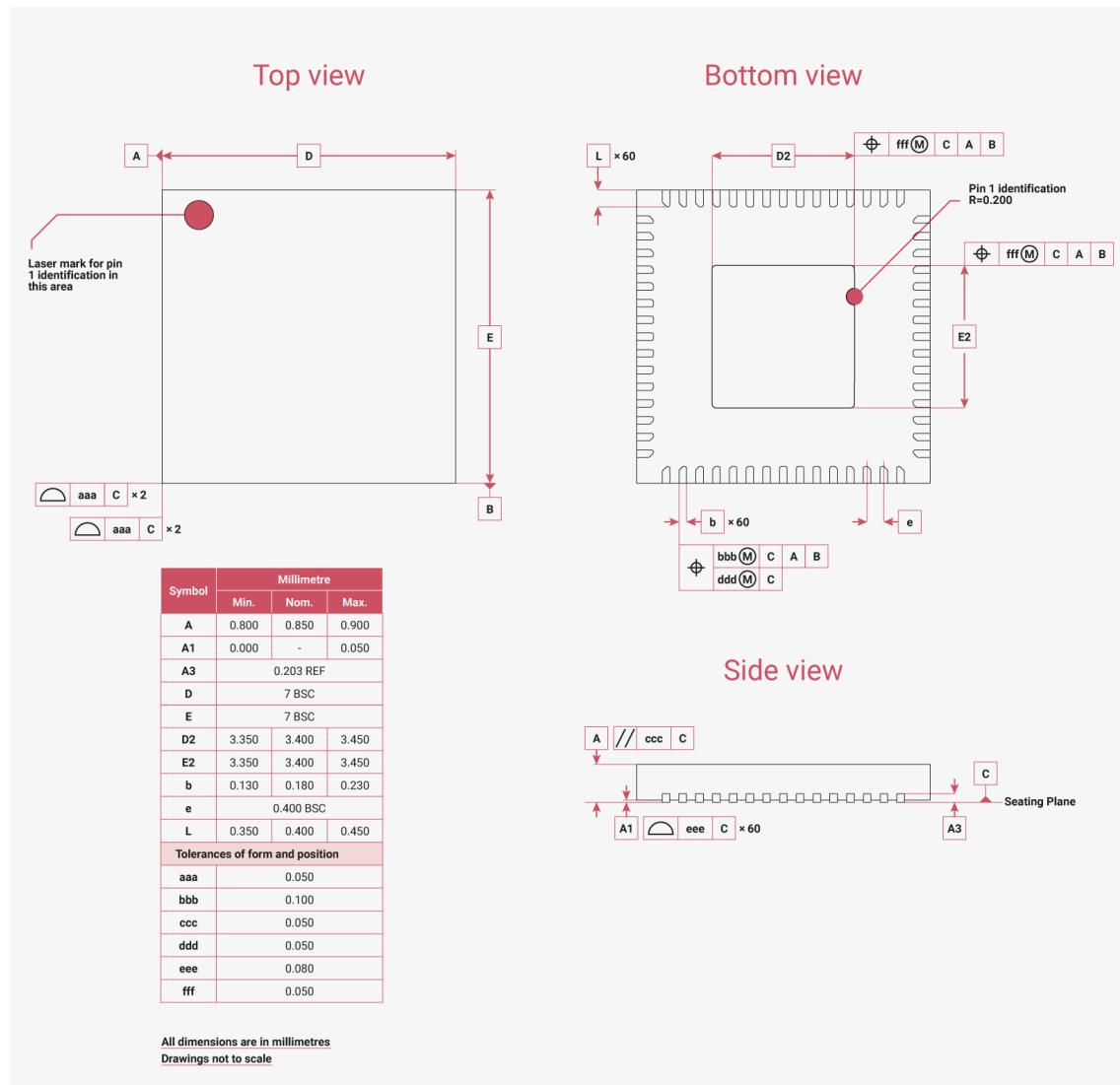
位	描述	类型	复位
1:0	<b>LOCK_S</b> : 该页面的安全访问锁定状态。采用温度计编码，锁定状态可通过编程OTP永久升级，从任一状态变更为权限更低的状态。软件亦可通过SW_LOCK寄存器临时提升锁定状态（直到下一次OTP复位）。	只读	-
	枚举值：		
	0x0 → READ_WRITE：页面可被安全软件完全访问。		
	0x1 → READ_ONLY：页面可被安全软件读取，但不可写入。		
	0x2 → 保留：禁止使用，行为等同于INACCESSIBLE。		
	0x3 → 不可访问：安全软件无法访问该页面。		

# 第14章。电气与机械

本节包含RP2350的物理及电气详细信息。

## 14.1. QFN-60封装

图 143。RP2350 QFN-60 封装的俯视图（左上）、底视图（右上）及侧视图（右下）



### ⓘ 注意

引线采用哑光锡（Sn）表面处理。退火在镀层后进行，温度为150°C，时间为1小时。引线镀层的最小厚度为8微米，中间层材料为CuFe2P（粗化铜Cu）。

### 14.1.1. 热特性

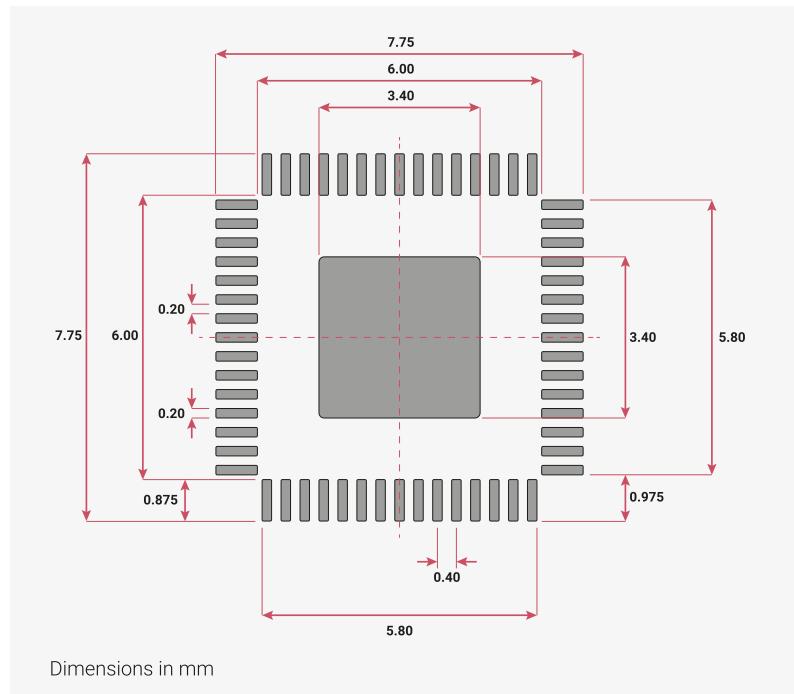
QFN-60封装的热特性见表1423。

表1423。 QFN-60  
封装的热  
性能数据。

器件	$\theta_{JA}$ (°C/W) - 静止 空气	$\theta_{JA}$ (°C/W) - 1m/s 强制空气流	$\theta_{JA}$ (°C/W) - 2m/s 强制空气流	$\theta_{JB}$ (°C/W)	$\theta_{JC}$ (°C/W)
RP2350A	40.542	31.99	30.264	12.588	14.315
RP2354A	待定	待定	待定	待定	待定

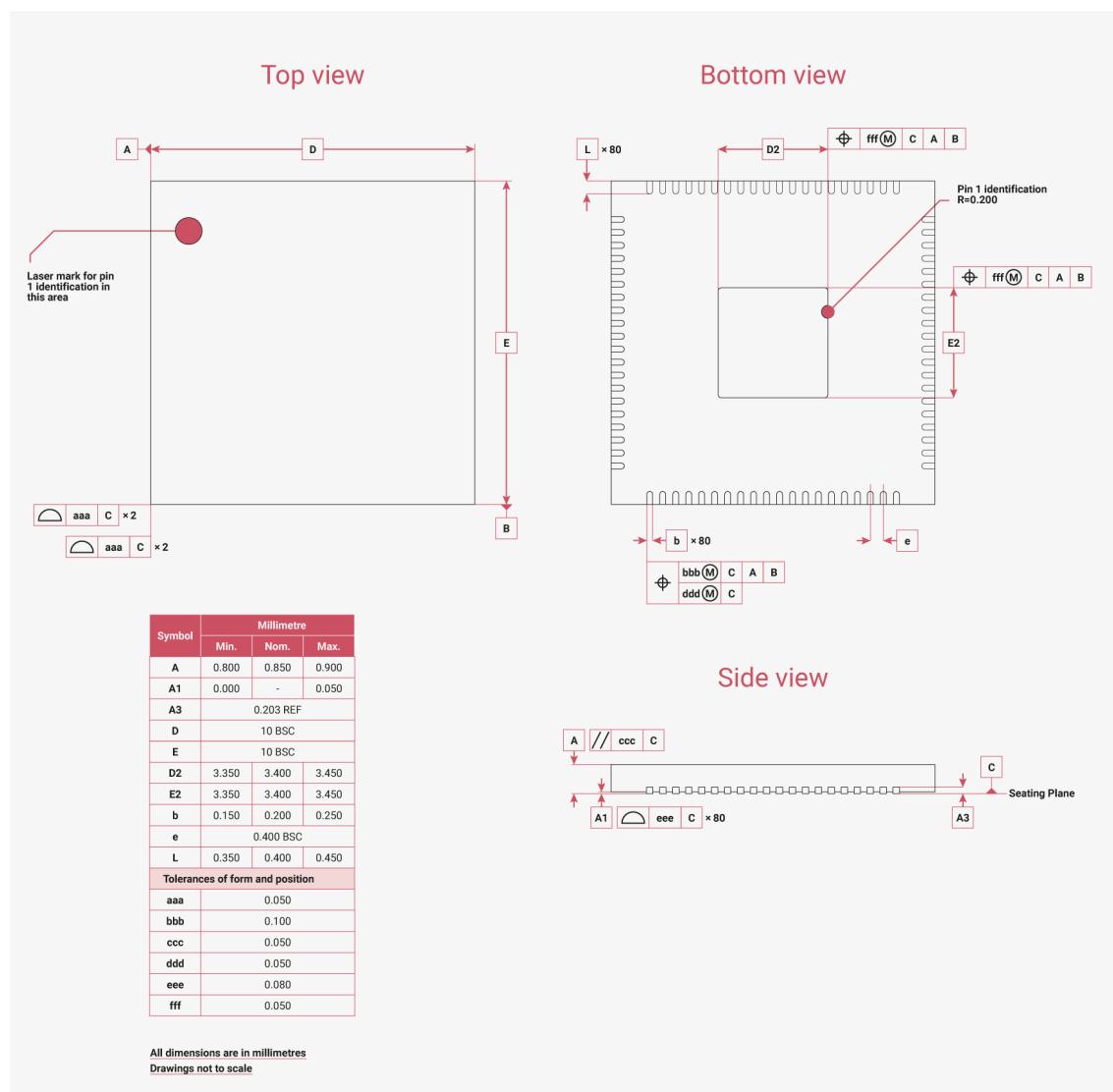
### 14.1.2. 推荐PCB封装尺寸

图144。  
RP2350 QFN-60封  
装推荐PCB封  
装尺寸



## 14.2. QFN-80封装

图145。RP2350 QFN  
N-80封装的顶视图  
图(左上)、  
侧视图(右下)  
及底视图(右上)  
)



### ① 注意

引线采用哑光锡(Sn)表面处理。退火在镀层后进行，温度为150°C，时间为1小时。引线镀层的最小厚度为8微米，中间层材料为CuFe2P(粗化铜Cu)。

### 14.2.1. 热特性

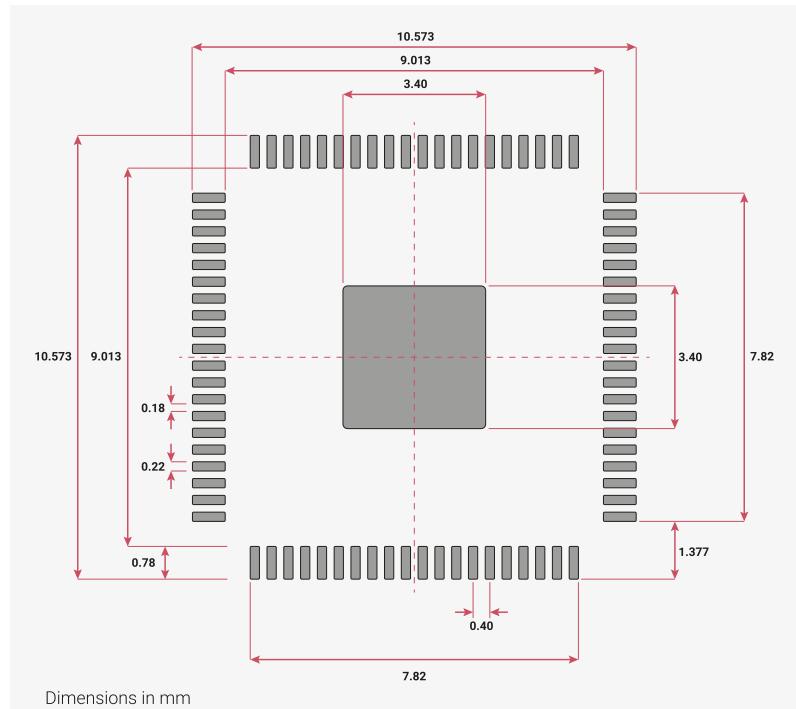
QFN-80封装的热特性详见表1424。

表1424。QFN-80  
封装的  
热数据。

器件	$\theta_{JA}$ (°C/W) - 静止 空气	$\theta_{JA}$ (°C/W) - 1m/s 强制空气流	$\theta_{JA}$ (°C/W) - 2m/s 强制空气流	$\theta_{JB}$ (°C/W)	$\theta_{JC}$ (°C/W)
RP2350B	待定	待定	待定	待定	待定
RP2354B	待定	待定	待定	待定	待定

### 14.2.2. 推荐PCB封装尺寸

图 146。  
RP2350 QFN-80封  
装推荐PCB封  
装尺寸



### 14.3. 封装内闪存

RP2354A 和 RP2354B 具备 2 MB 内部闪存。在其他所有方面，包括引脚排列，它们与无闪存版本的 RP2350A 和 RP2350B 完全相同。它们采用相同的 QFN-60 (RP2354A) 和 QFN-80 (RP2354B) 封装。RP2354 器件包含两层叠加硅芯片：

- 与无闪存版本相同的 RP2350 硅芯片
- 一颗 Winbond W25Q16JVWI QSPI NOR 闪存

#### Winbond W25Q16JVWI 数据手册

有关 RP2354 中使用的 W25Q16JVWI 器件的详细信息，详见：

[www.winbond.com/hq/product/code-storage-flash-memory/serial-nor-flash/?\\_\\_locale=en&partNo=W25Q16JV](http://www.winbond.com/hq/product/code-storage-flash-memory/serial-nor-flash/?__locale=en&partNo=W25Q16JV)

RP2350 芯片上的六个专用 QSPI 引脚 (**CSn**、**SCK** 及 **SD0** 至 **SD3**) 连接至内部闪存芯片及外部封装引脚。这使它们在以下方面表现类似于无闪存的 RP2350 设备：

- QSPI 的 **CSn** 引脚可在复位或上电时被拉低，以选择 BOOTSEL 模式。
  - 此操作仅无害地选择内部闪存芯片，但不会向其发送任何命令。
- 选择 BOOTSEL 时，QSPI 的 **SD1** 引脚可被拉高以选用 UART 启动。
  - UART 的 TX 信号通过 **SD2** 引脚输出，RX 信号通过 **SD3** 引脚输入，详见第 5.8 节。
  - 即使片选信号拉低，且 SCK 线上无信号变化，内部闪存芯片在 **SD0** 至 **SD3** 引脚保持高阻抗状态，因此可在 UART 启动过程中持续保持 **CSn** 信号为低电平。
- 内部闪存可通过 USB BOOTSEL 模式下的 UF2 拖放下载方式进行编程。
- 可以通过将第二个 QSPI 设备连接到 QSPI 引脚及来自 Bank 0 GPIO 的二级片选信号，实现外部附加
  - 该设备可用于扩展闪存容量或外部 QSPI RAM

- 有关RP2350 QSPI存储器接口及其功能的更多详细信息，详见第12.14节

通过将RP2350芯片保持在复位状态（通过RUN引脚，低电平有效复位），并由外部编程器驱动QSPI信号向芯片输入，可对内部闪存芯片进行外部编程

内部闪存由QSPI\_IOVDD电源输入供电该电压必须维持在2.7至3.6伏范围内。设计去耦电路及PCB布局时，应充分考虑此电源引脚上的高频电流增加

W25Q16JVWI的最大QSPI时钟频率为133 MHz。详细时序与交流参数请参阅W25Q16JVWI数据手册

如不需要从外部访问RP2350的QSPI总线，应尽量缩短连接至PCB上QSPI封装引脚的线路长度，以避免不必要的电磁发射及QSPI总线的电容负载。

W25Q16JVWI上的PADRESETB复位输入未连接至任何外部封装引脚或RP2350芯片内部的任何信号，故无法通过硬件方式复位闪存芯片。RP2350芯片退出复位状态时，会通过发出固定的XIP退出序列初始化闪存芯片，恢复其至串行命令状态，为执行就地运行配置做好准备，方式与外部闪存设备一致。

## 14.4. 封装标记

RP2350提供7×7 mm QFN-60封装及10×10 mm QFN-80封装，封装上标注如下信息：

- 引脚1标记点
- 标志
- 部件型号
- 日期代码（周）
- 硅片批号
- 日期代码（年份）

部件编号由以下部分组成：

- 设备名称“RP2350”
- 封装类型，‘A’表示QFN-60，‘B’表示QFN-80
- 封装版本“0”
- 晶圆版本‘A2’、‘A3’或‘A4’

有关晶圆版本之间差异的汇总，请参见附录C。

## 14.5. 存储条件

为保证裸露RP2350器件的存储和使用寿命，请遵守JEDEC J-STD (020E和033D) 标准。

RP2350 QFN-60封装被归类为湿敏等级1 (MSL1)。QFN-80的湿敏等级尚在评估中，相关细节将在未来数据手册更新中公布。

所有RP2350器件应存放于温度低于30°C且相对湿度不超过85%的环境中。

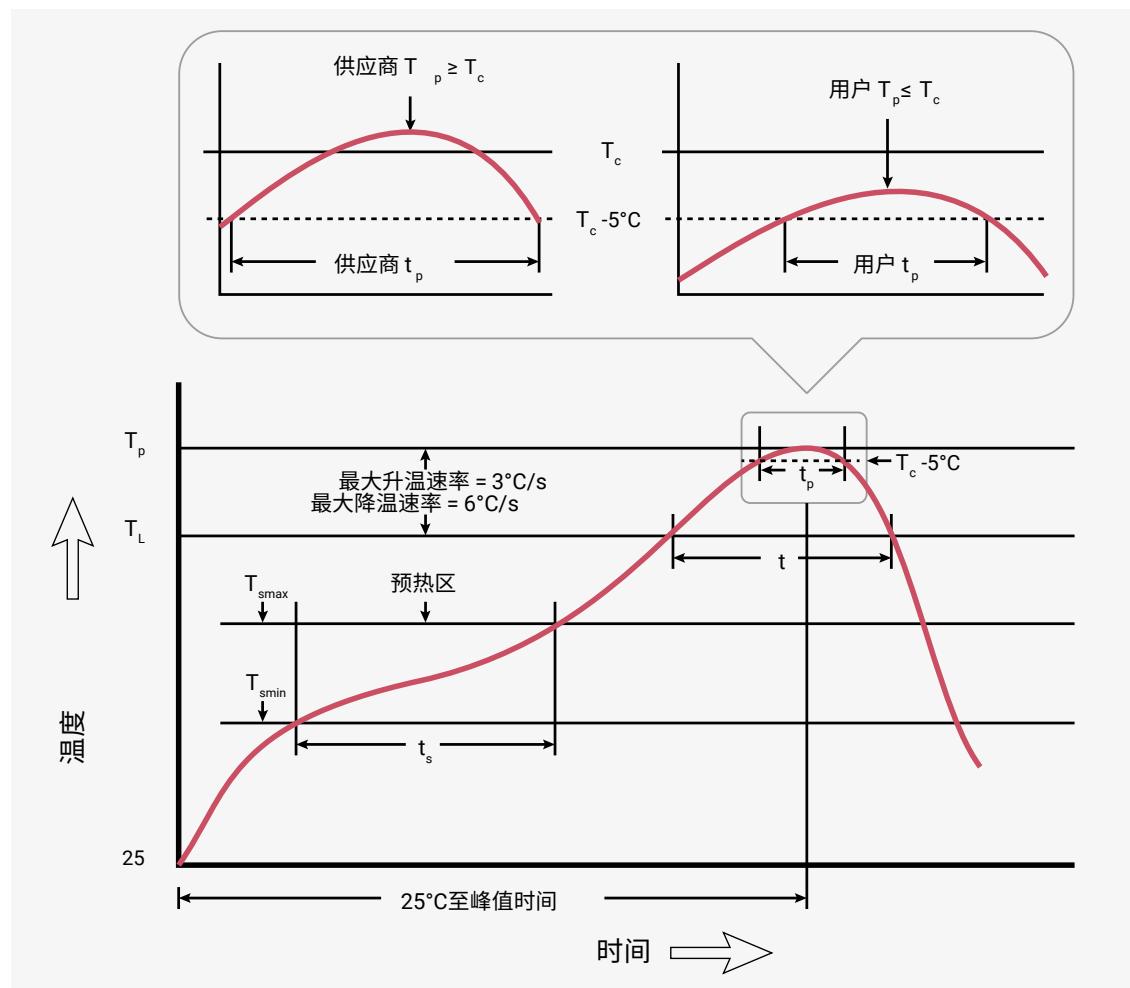
## 14.6. 焊接曲线

RP2350为无铅器件，熔点温度( $T_p$ )为260°C。

所有温度均指封装中心，测量位置为封装正面朝上的封装主体表面。

组装回流（倒置芯片方向）。如果元件以不同方向回流（例如，投死芯片方向）， $T_p$ 应与倒置芯片方向的 $T_p$ 保持 $\pm 2^\circ\text{C}$ 以内，且仍需满足 $T_c$ 的要求；否则，必须调整曲线以满足后者要求。

图 147.  
分类概要  
(非按比例)



本文档中的回流曲线用于分类及预处理，非用于规定板级组装工艺曲线。实际板级组装曲线应依据具体工艺需求及板设计制定，且不得超过表 1425 所列参数。

表 1425. 焊接  
曲线参数

曲线特征	值
最低温度 ( $T_{smin}$ )	150°C
最高温度 ( $T_{smax}$ )	200°C
时间 ( $t_s$ ) 从 ( $T_{smin}$ 至 $T_{smax}$ )	60 - 120 秒
升温速率 ( $T_L$ 至 $T_p$ )	最高 3°C/秒
液相线温度 ( $T_L$ )	217°C
维持温度高于 $T_L$ 的时间 ( $t_L$ )	60 至 150 秒
峰值封装温度 ( $T_p$ )	260°C
分类温度 ( $T_c$ )	260°C
在指定分类温度 ( $T_c$ ) $\pm 5^\circ\text{C}$ 范围内的时间 ( $t_p$ )	30 秒
降温速率 (从 $T_p$ 至 $T_L$ )	最大 6°C/秒
从 25°C 升至峰值温度的时间	最长 8 分钟

## 14.7. 合规性

RP2350 QFN-60 符合湿度敏感等级 1 的要求。RP2350 QFN-80 的湿度敏感等级合规性尚未完全确认，相关详情将于未来数据表更新中公布。

RP2350 符合 REACH 高关注度物质（SVHC）欧盟 ECHA 指令的要求。

RP2350 符合 2011/65/EU 欧盟 RoHS 指令及 2015/863 欧盟修订指令关于受控环境相关物质的要求与标准。

Raspberry Pi Ltd 对 RP2350 进行了以下封装级可靠性认证：

- 按 JESD22-A104 进行温度循环测试
- 按 JESD22-A110 进行高加速应力测试（HAST）
- 按 JESD22-A103 进行高温存储寿命测试（HTSL）
- 按 JESD22-A113 进行湿气敏感等级（MSL）测试

同时进行了以下硅片级可靠性认证：

- 按 JESD22-A108F 进行高温运行寿命测试（HTOL）

### ① 注意

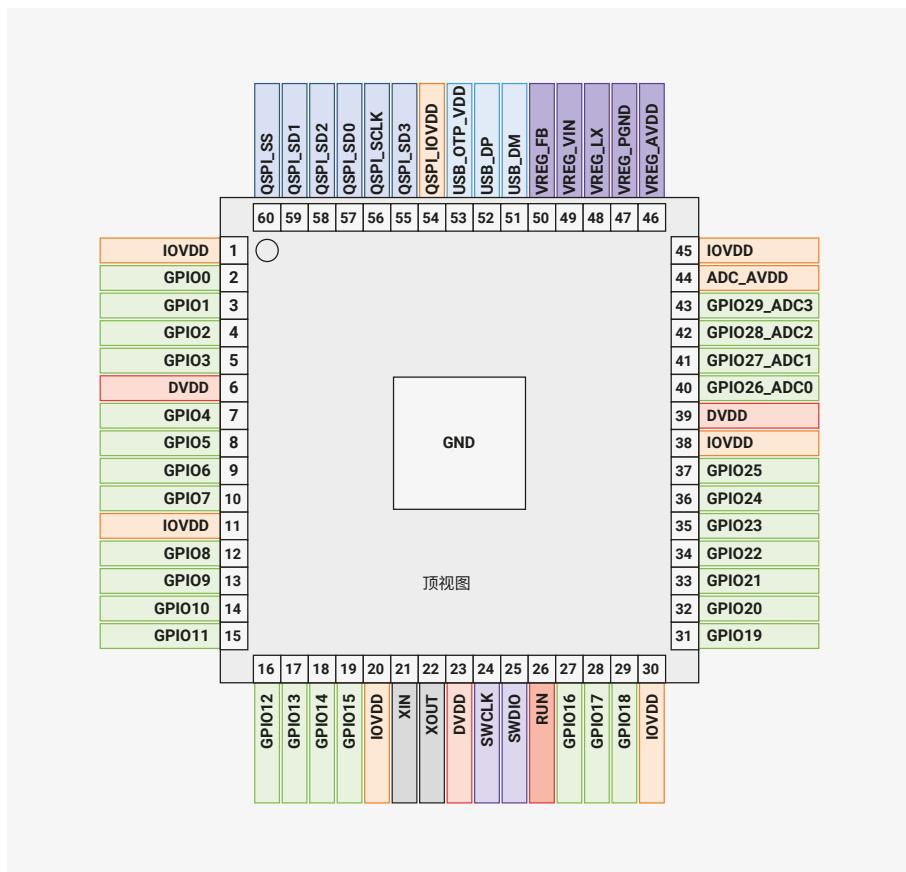
未执行锡须测试。RP2350 在 QFN-60 和 QFN-80 封装中为仅底部端接器件，故不适用 JEDEC 标准（JESD201A）。

## 14.8. 引脚排列

### 14.8.1. 引脚位置

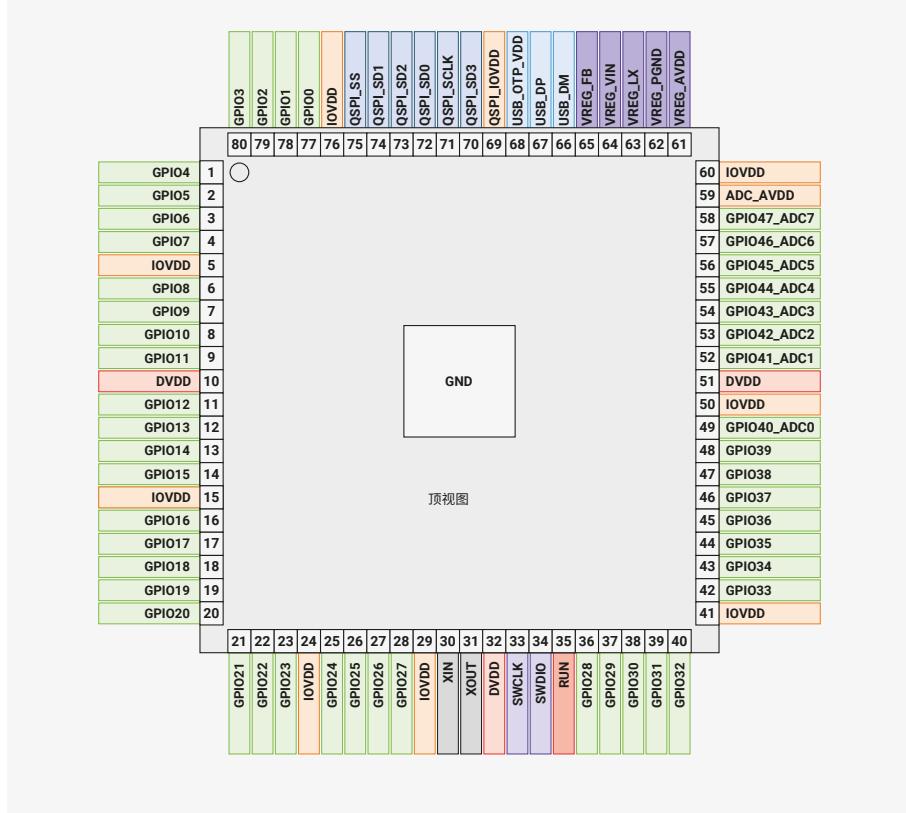
#### 14.8.1.1. QFN-60 (RP2350A)

图148。 RP2350  
QFN-60引脚布局  
7x7毫米



#### 14.8.1.2. QFN-80 (RP2350B)

图149。 RP2350  
QFN-80引脚布局  
10x10毫米



## 14.8.2. 引脚定义

### 14.8.2.1. 引脚类型

在下表（表1427）中，定义了以下引脚类型。

表1426. 引脚类型

引脚类型	方向	描述
数字输入	仅输入	标准数字。可编程上拉、下拉、斜率控制、施密特触发器及驱动强度。默认驱动强度为4 mA。
数字IO	双向	
数字输入（容错）	仅输入	容错数字。这些引脚称为容错型，意指在引脚电压低于3.63 V且I <sub>OVDD</sub> 为0 V时，流入引脚的电流极小。此外，若I <sub>OVDD</sub> 供电为3.3 V，则其可承受高达5.5 V的电压。该引脚具备增强的ESD保护功能。可编程上拉、下拉、斜率控制、施密特触发器及驱动强度。默认驱动强度为4 mA。
数字输入输出（FT）	双向	
数字输入输出 / 模拟	双向（数字），输入（模拟）	标准数字和ADC输入。可编程上拉、下拉、斜率控制、施密特触发器及驱动强度。默认驱动强度为4 mA。
USB 输入输出	双向	这些引脚用于USB，包含符合USB规范的内部上拉电阻和下拉电阻。USB操作需要外部27Ω串联电阻。
模拟（XOSC）		振荡器输入引脚，用于连接12 MHz晶体。或由方波驱动XIN引脚。

### 14.8.2.2. 引脚列表

表1427. GPIO引脚

名称	QFN-60编号	QFN-80编号	类型	电源域	复位状态	描述
GPIO0	2	77	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO1	3	78	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO2	4	79	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO3	5	80	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO4	7	1	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO5	8	2	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO6	9	3	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO7	10	4	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO8	12	6	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO9	13	7	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO10	14	8	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO11	15	9	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO12	16	11	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO13	17	12	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出
GPIO14	18	13	数字输入输出（FT）	I <sub>OVDD</sub>	下拉	用户输入输出

名称	QFN-60编号	QFN-80编号	类型	电源域	复位状态	描述
GPIO15	19	14	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO16	27	16	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO17	28	17	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO18	29	18	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO19	31	19	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO20	32	20	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO21	33	21	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO22	34	22	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO23	35	23	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO24	36	25	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO25	37	26	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO26_ADC0	40	-	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO27_ADC1	41	-	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO28_ADC2	42	-	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO29_ADC3	43	-	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO26	-	27	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO27	-	28	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO28	-	36	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO29	-	37	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO30	-	38	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO31	-	39	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO32	-	40	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO33	-	42	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO34	-	43	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO35	-	44	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO36	-	45	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO37	-	46	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO38	-	47	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO39	-	48	数字输入输出 (FT)	IOVDD	下拉	用户输入输出
GPIO40_ADC0	-	49	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO41_ADC1	-	52	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入

名称	QFN-60编号	QFN-80编号	类型	电源域	复位状态	描述
GPIO42_ADC2	-	53	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO43_ADC3	-	54	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO44_ADC4	-	55	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO45_ADC5	-	56	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO46_ADC6	-	57	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入
GPIO47_ADC7	-	58	数字输入/输出 / 模拟	IOVDD / ADC_AVDD	下拉	用户 IO 或 ADC 输入

表 1428. QSPI 引脚

名称	QFN-60编号	QFN-80编号	类型	电源域	复位状态	描述
QSPI_SD3	55	70	数字IO	QSPI_IOVDD	上拉	QSPI 数据
QSPI_SCLK	56	71	数字IO	QSPI_IOVDD	下拉	QSPI 时钟
QSPI_SD0	57	72	数字IO	QSPI_IOVDD	下拉	QSPI 数据
QSPI_SD2	58	73	数字IO	QSPI_IOVDD	上拉	QSPI 数据
QSPI_SD1	59	74	数字IO	QSPI_IOVDD	下拉	QSPI 数据
QSPI_SS	60	75	数字IO	QSPI_IOVDD	上拉	QSPI 芯片选择 / USB BOOTSEL

表 1429. 晶振引脚

名称	QFN-60编号	QFN-80编号	类型	电源域	描述
XIN	21	30	模拟 (XOSC)	IOVDD	晶体振荡器 XIN也可以由方波信号驱动。
XOUT	22	31	模拟 (XOSC)	IOVDD	晶体振荡器

表 1430  
杂项引脚

名称	QFN-60编号	QFN-80编号	类型	电源域	复位状态	描述
RUN	26	35	数字输入 (容错)	IOVDD	上拉	芯片使能 / reset_n
SWCLK	24	33	数字输入 (容错)	IOVDD	上拉	串行线调试时钟
SWDIO	25	34	数字输入输出 (FT)	IOVDD	上拉	串行线调试数据

表 1431. USB 引脚

名称	QFN-60编号	QFN-80编号	类型	电源域	描述
USB_DP	52	67	USB 输入输出	USB OTP_VDD	USB 数据正极 USB操作 需27Ω串联电阻

名称	QFN-60编号	QFN-80编号	类型	电源域	描述
USB_DM	51	66	USB 输入输出	USB OTP_VDD	USB数据负极。USB操作需27Ω串联电阻

表1432。电源引脚

名称	QFN-60引脚编号	QFN-80引脚编号	描述
DVDD	6, 23, 39	10, 32, 51	核心电源
IOVDD	11, 20, 30, 38, 45, 54	5, 15, 24, 29, 41, 50, 60, 76	IO电源
QSPI_IOVDD	54	69	QSPI IO电源
USB OTP_VDD	53	68	USB与OTP电源
ADC_AVDD	44	59	ADC电源
VREG_AVDD	46	61	电压调节器模拟供电
VREG_PGND	47	62	电压调节器地线
VREG_LX	48	63	电压调节器开关输出（连接至电感）
VREG_VIN	49	64	电压调节器输入供电
VREG_FB	50	65	电压调节器反馈输入
GND	-	-	通过中央裸露焊盘接地

## 14.9. 电气规格

以下电气规格基于指定温度、电压范围及工艺变异的特性测试获得，除非规格标注为“模拟”。在此情况下，数据仅供参考，且不保证准确性。

### 14.9.1. 绝对最大额定值

超过下表所列绝对最大额定值的应力可能导致器件永久损坏。这些仅为应力额定值，不代表器件的功能性操作范围。

表1433。绝对最大额定值

参数	符号	条件	最小值	最大值	单位	备注
核心供电 (DVDD) 电压	DVDD		-0.5	1.21	V	
I/O供电 (IOVDD) 及 QSPI 供电 (QSPI_IOVD) 电压	IOVDD		-0.5	3.63	V	

参数	符号	条件	最小值	最大值	单位	备注
IO电压 (标准)	V <sub>引脚</sub>		-0.5	IOVDD + 0.5	V	
IO电压 (FT)	V <sub>引脚_FT</sub>	IOVDD=3.3V	-0.5	5.5	V	IOVDD必须存在
		IOVDD=2.5V	-0.5	4.2	V	
		IOVDD=1.8V	-0.5	3.63	V	
		IOVDD=0V	-0.5	3.63	V	
结温			-40	125	°C	
储存 温度				150	°C	

### 14.9.2. 静电放电性能

表 1434. 除非另有说明，所有引脚的ESD性能

参数	符号	最大值	单位	备注
人体模型	HBM	2	kV	符合JEDEC规范 JS-001-2012 (2012年4月)
人体模型 仅数字 (FT) 引脚	HBM	4	kV	符合JEDEC规范 JS-001-2012 (2012年4月)
带电器件模型	CDM	500	V	符合 JESD22-C101E (2009年12月)

### 14.9.3. 热性能

表 1435. 热性能

参数	符号	最小值	典型值	最大值	单位	备注
环境 温度	T <sub>c</sub>	-40		85	°C	

### 14.9.4. IO电气特性

表 1436. 数字IO特性—标准及FT  
T<sub>c</sub> 除非另有说明。本表中/IOVDD在适当情况下亦指QSP1/IOVDD

参数	符号	条件	最小值	最大值	单位	备注
引脚输入 泄漏 电流	I <sub>IN</sub>			1	μA	
输入电压 高 (标准 IO)	V <sub>IH</sub>	IOVDD=1.8V	0.65 * IOVDD	IOVDD + 0.3	V	
		IOVDD=2.5V	1.7	IOVDD + 0.3	V	
		IOVDD=3.3V	2	IOVDD + 0.3	V	

参数	符号	条件	最小值	最大值	单位	备注
输入电压高 (FT)	$V_{IH}$	IOVDD=1.8V	0.65 * IOVDD	3.63	V	IOVDD必须供电以承受高于3.63V的输入电压
		IOVDD=2.5V	1.7	4.2	V	
		IOVDD=3.3V	2	5.5	V	
输入电压低	$V_{IL}$	IOVDD=1.8V	-0.3	0.35 * IOVDD	V	
		IOVDD=2.5V	-0.3	0.7	V	
		IOVDD=3.3V	-0.3	0.8	V	
输入迟滞电压	$V_{HYS}$	IOVDD=1.8V	0.1 * IOVDD		V	施密特触发器已启用
		IOVDD=2.5V	0.2		V	
		IOVDD=3.3V	0.2		V	
输出电压高	$V_{OH}$	IOVDD=1.8V	1.24	IOVDD	V	$I_{OH} = 2、4、8 或 12mA$ , 取决于设置
		IOVDD=2.5V	1.78	IOVDD	V	
		IOVDD=3.3V	2.62	IOVDD	V	
输出电压低	$V_{OL}$	IOVDD=1.8V	0	0.3	V	$I_{OL} = 2, 4, 8 或 12mA$ 取决于设置
		IOVDD=2.5V	0	0.4	V	
		IOVDD=3.3V	0	0.5	V	
上拉电阻	$R_{PU}$	IOVDD=1.8V	32	106	kΩ	
		IOVDD=2.5V	42	123	kΩ	
		IOVDD=3.3V	32	86	kΩ	
下拉电阻	$R_{PD}$	IOVDD=1.8V	35	189	kΩ	
		IOVDD=2.5V	49	180	kΩ	
		IOVDD=3.3V	36	113	kΩ	
最大总 IOVDD 电流	$I_{IOVDD\_MAX}$			100	mA	所有GPIO引脚输出电流总和
最大总 QSPI_IOVDD 电流	$I_{QSPI\_IOVDD\_MAX}$			20	mA	所有QSPI引脚输出电流总和
最大总 VSS 电流 由GPIO引脚产生的 (IOVSS)	$I_{IOVSS\_MAX}$			100	mA	所有流入GPIO引脚的电流总和
由QSPI引脚产生的最大总 VSS电流 (QSPI_IOVSS)	$I_{QSPI\_IOVSS\_MAX}$			20	mA	汇入QSPI引脚的总电流

表1437. USB IO 特性

参数	符号	最小值	最大值	单位	备注
引脚输入漏电流	$I_{IN}$		1	$\mu A$	
单端输入 高电平电压	$V_{IHSE}$	2		V	
单端输入 低电平电压	$V_{ILSE}$		0.8	V	
差分输入 高电平电压	$V_{IHDIFF}$	0.2		V	
差分输入 低电平电压	$V_{ILDIFF}$		-0.2	V	
输出电压 高	$V_{OH}$	2.8	USB_OTG_VDD	V	
输出电压 低	$V_{OL}$	0	0.3	V	
上拉电阻 - RPU2	$R_{PU2}$	0.873	1.548	kΩ	
上拉电阻 - RPU1和RPU2	$R_{PU1\&2}$	1.398	3.063	kΩ	
下拉 电阻	$R_{PD}$	14.25	15.75	kΩ	

表1438. ADC  
特性

参数	符号	最小值	典型值	最大值	单位	备注
ADC输入 电压范围	$V_{PIN\_ADC}$	0		ADC_AVDD	V	
有效 位数	ENOB	9	9.5		位	
分辨位数				12	位	
ADC输入 阻抗	$R_{IN\_ADC}$	100			kΩ	

表1439。振荡器  
引脚特性

参数	符号	最小值	典型值	最大值	单位	备注
输入 频率	$f_{osc}$	1	12	50	MHz	有关PLL施 加的限 制，请参见 第8.6.3节 。  有关USB和 UART引导 程序施加的 限制，请参见 第5.2.8.1节。

参数	符号	最小值	典型值	最大值	单位	备注
输入电压 高电平	$V_{IH}$	$0.65*IOVDD$		$IOVDD + 0.3$	V	方波 输入。仅限 $XIN$ 端。 $XOUT$ 悬空
输入电压 低	$V_{IL}$	0		$0.35*IOVDD$	V	方波 输入。仅限 $XIN$ 端。 $XOUT$ 悬空

**注意**

默认情况下，USB引导模式依赖12MHz输入信号。然而，OTP可配置以在USB引导模式期间覆盖XOSC和PLL设置。详情请参见第13.10节。

有关振荡器的详细信息，请参见第8.2节；有关晶体使用，请参见RP2350硬件设计中的最简设计示例。

表1440。SWCLK  
引脚特性

参数	符号	最小值	典型值	最大值	单位	备注
SWCLK 输入 频率	$f_{SWCLK}$	0	10	50	MHz	有关 $SWCLK$ 引脚 定义，请参 见表1430。

主机至目标的数据应通过  $SWDIO$  引脚，且与  $SWCLK$  中心对齐传输。目标至主机的数据在  $SWDIO$  引脚上于  $SWCLK$  上升沿转换。

**注意**

RP2350内部SWD逻辑（位于SW-DP）在最高50 MHz下可稳定运行，但外部SWD信号的信号完整性仍可能存在挑战。

若出现如SW-DP写数据奇偶校验错误等SWD操作不稳定，应降低  $SWCLK$  频率。

除  $SWDIO$  和  $SWCLK$  外，务必直接连接SWD探针与RP2350之间的地线。请尽量缩短探针与RP2350之间的导线长度，并在高频率条件下避免采用多点串联布线。

#### 14.9.4.1. GPIO输出电压规格的解读

RP2350的GPIO具备四种不同的输出驱动强度，标称为2mA、4mA、8mA及12mA模式。这些数值并非严格限制，也不保证GPIO总是输出（或吸收）所设定的电流。

GPIO输出或吸收的电流视所连接负载而定。GPIO会尝试将输出电平驱动至IOVDD电平（逻辑0时为0V），但其可输出的电流有限，且依赖所选驱动强度。

因此，负载电流越大，针脚电压越低。当GPIO输出电流过大时，电压将下降到无法被连接设备识别为逻辑1的水平。

表1436的输出规格量化了在从引脚抽取指定电流时电压预期降低的幅度。

输出高电压 ( $V_{OH}$ ) 定义为在以特定驱动强度驱动输出引脚为逻辑1时的最低输出电压；例如，在4mA驱动强度模式下，针脚输出4mA电流。输出低电压与此类似，但指的是驱动为逻辑0时的电压。

此外，从IOVDD电源组（主要包括GPIO及QSPI引脚）输出（即输出为高电平时）的电流总和不得超过  $I_{IOVDD\_MAX}$ 。同理，所有下拉电流总和（即输出为低电平时）不得超过  $I_{IOVSS\_MAX}$ 。

图150。GPIO输出的典型电流-电压曲线。

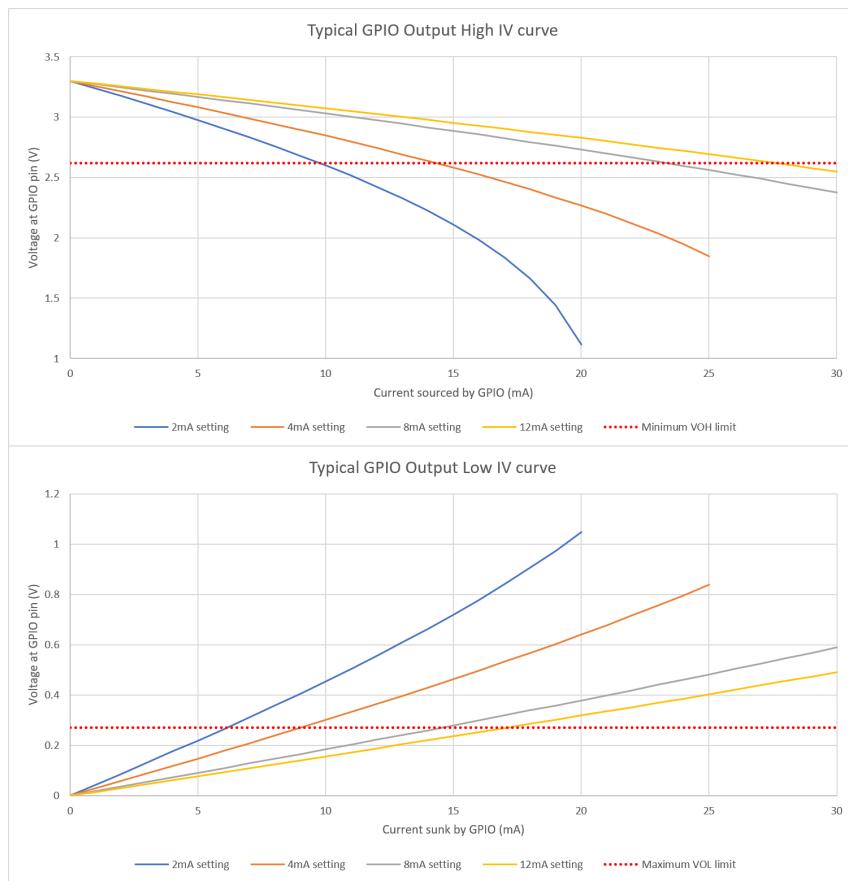


图150显示了随着引脚负载电流增加，输出电压的变化情况。可清晰见到不同驱动强度的影响；在给定电流下，驱动强度越高，输出电压越接近 $\text{IOVDD}$ （或 $0\text{V}$ ）。最小 $V_{\text{OH}}$ 和最大 $V_{\text{OL}}$ 限值以红色标示。

可见每种驱动强度对应的额定电流下，电压均在允许范围内，表明该器件可驱动更大电流且仍满足 $V_{\text{OH}}/V_{\text{OL}}$ 规格。此为室温下的典型器件，但器件存在差异，部分器件电压或将更接近极限值。

若应用不需严格电压控制，可从GPIO源出或吸入超过所选驱动强度设置的电流。但需通过实验验证在具体应用中此操作的安全性。

#### 14.9.5. 电源供应

表 1441. 电源供应规格

电源	供应	最小值	典型值	最大值	单位
<b>IOVDD<sup>a</sup></b>	数字IO	1.62	1.8 / 3.3	3.63	V
<b>QSPI_IOVDD</b> (仅限 RP2350) <sup>a</sup>	数字IO	1.62	1.8 / 3.3	3.63	V
<b>QSPI_IOVDD</b> (仅限 RP2354)	数字IO	2.97	3.3	3.63	V
<b>DVDD<sup>b</sup></b>	数字内核	1.05	1.1	1.16	V
<b>VREG_VIN</b>	电压调节器	2.7	3.3	5.5	V
<b>VREG_AVDD</b>	电压调节器	3.135	3.3	3.63	V

电源	供应	最小值	典型值	最大值	单位
USB_OTP_VDD	USB PHY 与 OTP	3.135	3.3	3.63	V
ADC_AVDD <sup>c</sup>	ADC	1.62	3.3	3.63	V

<sup>a</sup> 若 IOVDD < 2.5V，应相应调整 GPIO VOLTAGE\_SELECT 寄存器。详见第 6.1 节。

<sup>b</sup> 短期瞬态应控制在 ±100mV 以内。

<sup>c</sup> 电压低于 2.97V 时，ADC 性能将会受损。

### ● 注意

RP2350 内含 3.3V 内部闪存设备，故 QSPI\_IOVDD 必须为 3.3V。此外，若 QSPI 引脚用于连接额外闪存或 PSRAM 设备，则 IOVDD 必须为 3.3V，因为此情况下 GPIO 用作 QSPI 片选信号。

## 14.9.6. 核心电压调节器

表 1442. 电压  
稳压器  
规格

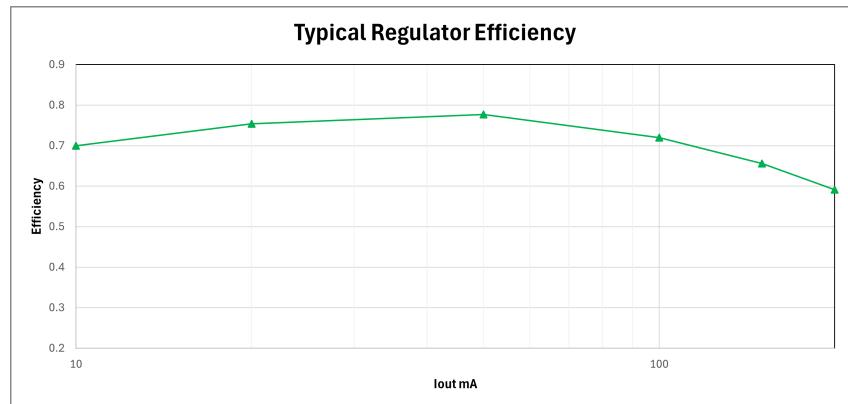
参数	描述	最小值	典型值	最大值	单位
$V_{OUT}$ (正常模式)	调节输出 电压范围 (正常模式)	0.55	1.1	3.3	V
$V_{OUT}$ (低功耗模式)	调节输出电压范 围 (低功耗模式)	0.55	1.1	1.3	V
$\Delta V_{OUT}$ (正常模式)	偏离设定值的电 压 (正常模式)	-3		+3	所选输出电 压的百分比
$\Delta V_{OUT}$ (低功耗模式)	偏离设定值的电 压 (低功耗模式)	-9		+9	所选输出电 压的百分比
$I_{MAX}$ (正常模式)	输出电流 (正常模式)			200	mA
$I_{MAX}$ (低功耗模式)	输出电流 (低功耗模式)			1	mA
$I_{LIMIT}$ (正常模式启动)	电流限制 (正常模式 启动)		240	300	mA
$I_{LIMIT}$ (正常模式)	电流限制 (正常模式)	260	500	800	mA
$I_{LIMIT}$ (低功耗模式)	电流限制 (低 功耗模式)	5		25	mA
$V_{OUT\_OK}$ 阈值 · 断言 阈值	$V_{OUT\_OK}$ 断言 阈值	87	90	93	所选输出电 压的百分比

参数	描述	最小值	典型值	最大值	单位
VOUT_OK_阈值·取消断言 阈值	VOUT_OK 取消 断言 阈值	84	87	90	所选输出电 压的百分比
f开关	开关 频率		3		MHz
效率 ( $V_{out}=1.1V$ )	负载=10mA, VR EG_VIN=2.7V		74		%
	负载=10mA, VR EG_VIN=3.3V		70		%
	负载=10mA, VREG_VIN=5.5V		59		%
	负载=100mA, VREG_VIN=2.7V		70		%
	负载=100mA, VREG_VIN=3.3V		72		%
	负载=100mA, VREG_VIN=5.5V		72		%
	负载=200mA, VREG_VIN=2.7V		70		%
	负载=200mA, VREG_VIN=3.3V		59		%
	负载=200mA, VREG_VIN=5.5V		63		%

### ● 警告

$V_{out}$  可以超过最大核心电源 ( $DVDD$ ) 的电压。虽然存在防止意外发生的电压限制，但该限制可通过软件控制予以禁用。  
○ 为确保可靠运行，  $DVDD$  不应超过其最大电压额定值。

图151。典型稳压器效率， $V_{out}=1.1V$ ,  $V_{RE}$   
 $G\_VIN=3.3V$ 。



### 14.9.7. 功耗

### 14.9.7.1. 外设功耗

基线读数是在仅激活时钟源及必要外设（BUSCTRL、BUSFAB、VREG、复位、只读存储器、SRAM），且相应位于 [WAKE\\_E\\_N0/WAKE\\_EN1](#) 寄存器中的情况下采集。时钟设定为默认值。

通过在 [WAKE\\_EN0/WAKE\\_EN1](#) 寄存器中启用该外设的所有时钟源，依次激活各外设。

电流消耗为启用外设时钟后电流的增量。

表1443。基线功耗

外设	典型DVDD电流消耗 ( $\mu\text{A}/\text{MHz}$ )
DMA	2.6
I2C0	3
I2C1	3.6
IO + 封装引脚	24.5
PWM	9.9
SIO	2
SHA256	0.1
SPI0	1.7
SPI1	1.4
定时器 0	0.8
定时器 1	0.6
TRNG	0.8
UART0	2.6
UART1	3.6
看门狗	1.1
XIP	37.6

由于具有固定的48MHz参考时钟以及可变的系统时钟输入，ADC和USBCTRL的功耗不会随系统时钟线性变化（而其他仅包含系统和/或外设时钟输入的外设功耗则呈线性变化）。下表显示了ADC和USBCTRL模块在标准时钟设置下的绝对DVD D电流消耗：

表1444。ADC和USBCTRL的基线功耗

外设	典型DVDD电流消耗 (mA)
ADC	0.14
USBCTRL	1.25

### 14.9.7.2 低功耗状态下的功耗

表1445显示了低功耗状态 P1.0 → P1.7 的典型功耗。所有电压供应均为3.3V（DVDD除外，其由稳压器供电（在低功耗模式下）），环境温度为室温。

所有GPIO、[SWDIO](#)和[SWCLK](#)均内部下拉，且未外部连接。QSPI连接至W25Q16JVSSIQ闪存设备。USB PHY已断电，并且在进入低功耗状态前启用了[DP](#)和[DM](#)下拉。USB线缆仍连接至主机计算机。表中还显示了当[RUN](#)保持低电平时的功耗。此状态技术上并非低功耗（电压调节器处于正常开关模式），但为完整性而列出。

表1445。低功耗状态功耗

低功耗状态	VREG_VIN (μA)	VREG_AVDD (μA)	IOVDD (μA)	QSPI_IOVDD (μA)	ADC_IOVDD (μA)	USB OTP_VDD (μA)	总功率 (μW)
P1.0	128	0.5	11	22	1	3.5	548
P1.1	77	0.5	11	22	1	3.5	380
P1.2	79	0.5	11	22	1	3.5	380
P1.3	26	0.5	11	22	1	3.5	204
P1.4	120	0.5	11	22	1	3.5	520
P1.5	67	0.5	11	22	1	3.5	345
P1.6	68	0.5	11	22	1	3.5	350
P1.7	19	0.5	11	22	1	3.5	188
RUN=低	40	187	69	22	1	35	1170

#### 14.9.7.3 典型用户案例的功耗

下表详细列出了RP2350在各种典型应用场景下的功耗。所有测量均使用3.3V电源供电（DVDD除外，DVDD由电压调节器供电，设定为1.1V），测试环境为室温。

SWD和SWCLK未连接至外部。GPIO0和GPIO1连接至Raspberry Pi调试探针（UART），其余GPIO未连接（USB引导模式下，GPIO0和GPIO1也未连接）。QSPI连接至W25Q16JVSSIQ闪存器件，USB连接至主机。

hello\_serial、hello\_usb和hello\_adc是pico-examples中的基础示例程序，字符持续发送至串口控制台。

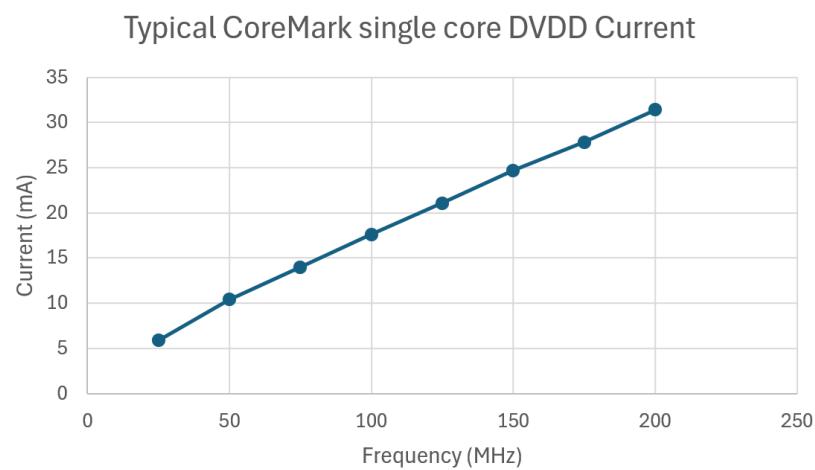
表1446。功耗

使用场景	条件	VREG_VIN (μA)	VREG_AVDD (μA)	IOVDD (μA)	QSPI_IOVDD (μA)	ADC_IOVDD (μA)	USB OTP_VDD (μA)	总功率 (mW)
USB引导模式	总线空闲（平均）	6530	220	437	22	1	375	25
	启动期间（峰值）						6050	
	UF2写入期间（平均值）						1280	
hello_serial		14690	216	506	22	1	62	51.1
hello_usb		14700	216	453	22	1	570	52.7
hello_adc		14680	216	506	22	142	62	51.6
CoreMark基准测试	单核@150MHz	11000	212	455	22	1	90	38.7

##### 14.9.7.3.1. 功耗与频率的关系

核心RP2350频率与DVDD电源电流之间存在关联。图152展示了典型RP2350设备在单核不同核心时钟频率下连续运行CoreMark基准测试的测量结果。

图152。典型RP  
2350设备运行  
CoreMark基准测试  
时的DVDD电流与核  
心频率关系



# 附录A：寄存器字段类型

## RP2040的变更

寄存器字段类型保持不变。

### 标准类型

#### 读写：

- 读/写
- 读操作返回寄存器值。
- 写操作更新寄存器值。

#### 只读：

- 只读
- 读操作返回寄存器值。
- 写操作被忽略。

#### 只写：

- 仅写
- 读操作返回0。
- 写操作更新寄存器值。

### 清除类型

#### SC:

- 自清除。
- 向SC字段中的某位写入1将触发事件，事件触发后该位自动清零。
- 向SC字段中的某位写入0则无任何动作。

#### WC:

- 写清除。
- 向WC字段中的某位写入1将使该位被写为0

- 向WC字段中的某位写入0不会产生任何效果
- 读操作返回寄存器值。

## FIFO 类型

这些字段用于从FIFO中读取和写入数据。相关寄存器提供FIFO的控制与状态信息。控制和状态寄存器无固定格式，因其针对每个FIFO接口均有所不同。

### RWF:

- 读/写FIFO
- 读取此字段将返回FIFO中的数据
  - 读取完成后，数据值将从FIFO中移除
  - 若FIFO为空，将返回默认值；默认值因具体FIFO接口而异
- 写入此字段的数据会被推入FIFO，FIFO满时的行为取决于具体接口
- 读写操作可能涉及不同的FIFO

### RF:

- 读FIFO
- 功能同读写FIFO，但仅限读取

### WF:

- 写FIFO
- 功能同读写FIFO，但仅限写入

# 附录 B：本文件中使用的单位

本数据表遵循NIST推荐的国际单位制（SI）单位使用标准，但内存和存储容量的用法除外。在此，采用前缀k（*kilo*）、M（*mega*）和G（*giga*）始终指代其标准十进制值附近的二进制幂次的惯例。此做法使本数据表与这些单位的惯用表达保持一致。

## 内存与存储容量

本数据表使用以下单位表示内存和存储容量：

- b：比特
  - kb：千比特， $2^{10}$  b = 1024 b
  - Mb：兆比特， $2^{20}$  b = 1,048,576 b
  - Gb：吉比特， $2^{30}$  b = 1,073,741,824 b
- B：字节，八比特，八位字节
  - kB：千字节， $2^{10}$  B = 1024 B
  - MB：兆字节， $2^{20}$  B = 1,048,576 B
  - GB：千兆字节， $2^{30}$  B = 1,073,741,824 B

比特是信息的最基本单位。比特的取值为真（1）或假（0）。

## 传输速率

传输速率的单位在量纲上是字节或比特与频率单位（如MHz）的乘积，因此适用标准国际单位制前缀：

- b/s：比特每秒
  - kb/s：千比特每秒，1000 b/s
  - Mb/s：兆比特每秒，1,000,000 b/s
  - Gb/s：千兆比特每秒，1,000,000,000 b/s
- B/s：字节每秒，8 b/s
  - kB/s：千字节每秒，1000 B/s
  - MB/s：兆字节每秒，1,000,000 B/s
  - GB/s：千兆字节每秒，1,000,000,000 B/s

## 物理量

以下单位表示物理量，如电压和频率：

- A：安培，电流单位，等于每秒一库仑

- mA：毫安， $10^{-3}$  A
- $\mu$ A：微安， $10^{-6}$  A
- Ω：欧姆，电阻或阻抗单位，等于每安培一伏特
  - MΩ：兆欧， $10^{-6}$  Ω
  - kΩ：千欧， $10^3$  Ω
  - mΩ：毫欧， $10^{-3}$  Ω
- V：伏特，电势差单位，等于每库仑一焦耳
  - kV：千伏， $10^3$  V
  - mV：毫伏， $10^{-3}$  V
  - $\mu$ V：微伏， $10^{-6}$  V
- Hz：赫兹，频率单位，等于每秒一个周期 ( $s^{-1}$ )
  - kHz：千赫兹， $10^3$  Hz
  - MHz：兆赫兹， $10^{-6}$  Hz
  - GHz：千兆赫兹， $10^9$  Hz
- F：法拉，电容量单位，库仑／伏特
  - mF：毫法拉， $10^{-3}$  F
  - $\mu$ F：微法拉， $10^{-6}$  F
  - nF：纳法拉， $10^{-9}$  F
  - pF：皮法拉， $10^{-12}$  F
- H：亨利，电感单位，伏秒／安培 ( $VsA^{-1}$ )
  - mH：毫亨利， $10^{-3}$  H
  - $\mu$ H：微亨利， $10^{-6}$  H
  - nH：纳亨利， $10^{-9}$  H
- s：秒，时间单位
  - ms：毫秒， $10^{-3}$  s
  - $\mu$ s：微秒， $10^{-6}$  s
  - ns：纳秒， $10^{-9}$  s
  - ps：皮秒， $10^{-12}$  s
- J：焦耳，能量或功的单位，等于一牛顿·米
- C：库仑，电荷单位
  - mC：毫库仑， $10^{-3}$  C
  - $\mu$ C：微库仑， $10^{-6}$  C
  - nC：纳库仑， $10^{-9}$  C
- m：米，长度或距离单位
  - mm：毫米， $10^{-3}$  m
- °C：摄氏度，温度单位
- W：瓦特，功率单位，等于一焦耳每秒 ( $J s^{-1}$ ) 或一伏安 (VA)

- mW：毫瓦特， $10^{-3}$  W
- $\mu$ W：微瓦特， $10^{-6}$  W
- nW：纳瓦特， $10^{-9}$  W

## 单位前缀

前述章节中使用的标准国际单位制前缀如下：

- G：吉，倍数为 $10^9$ （一十亿）
- M：兆，倍数为 $10^6$ （一百万）
- k：千， $10^3$ 次方（千）
- c：厘， $10^{-2}$ 次方（百分之一）
- m：毫， $10^{-3}$ 次方（千分之一）
- $\mu\mu$ ：微， $10^{-6}$ 次方（百万分之一）
- n：纳， $10^{-9}$ 次方（十亿分之一）
- p：皮， $10^{-12}$ 次方（万亿分之一）

内存和存储容量部分采用的常用二进制前缀如下：

- G：吉， $2^{30}$ 次方（约等于 $10^9$ 次方）
- M：兆， $2^{20}$ 次方（约等于 $10^6$ 次方）
- k：千， $2^{10}$ 次方（约等于 $10^3$ 次方）

这些常用二进制前缀对应IEC 60027-2标准中的以下前缀：

- Gi，吉比
- Mi，梅比
- Ki, kibi

## 数字分隔符

书写多位数字时，可能会插入逗号或空格以便于阅读：

- 1,000,000：一百万
- 1 000 000：一百万

数字中的逗号绝不表示小数点（基数点）。

# 附录 C：硬件版本历史

本附录总结了 RP2350 硬件版本之间的差异，称为 **steppings**。要确定未知设备的版本，请检查封装标记，如第14.4节所述。设备上的软件也可以读取 CHIP\_ID.REVISION 寄存器字段，或调用 `rp2350_chip_version()` SDK 函数。

在本附录中：

- “**fix**”一词指已完全解决的问题。
- “**mitigate**”一词指部分解决的问题，或被认为已完全解决但存在不可预测的潜在原因的问题，例如故障注入漏洞。
- “**update**”一词指**steppings**之间的其他任何差异。

本附录提供了高级概述；有关详细信息，请参阅附录E中的各个勘误条目。

## RP2350 A2

版本A2通过 CHIP\_ID.REVISION 值 `0x2`加以识别。

A2是RP2350的首个普遍可用版本，也是本数据手册中记载的最早版本。

## RP2350 A3

版本A3通过 CHIP\_ID.REVISION 值 `0x3`加以识别。

### 硬件变更

版本A3引入了以下硬件修复和缓解措施：

- **修复 RP2350-E3:** 在 QFN-60 封装中，`GPIO_NSmask` 错误控制了 `PADS` 寄存器。硬件现已将 `GPIO_NSmask` 重新映射至 QFN-60 封装中正确的 `PADS` 寄存器。
- **修复 RP2350-E9:** 当引脚输入启用时，Bank 0 GPIO 漏电流增加。电路已修改，以消除通过输入缓冲器的错误漏电通路。
- **缓解 RP2350-E12:** USB 状态信号同步不足的问题。Bootrom 使用的信号现已在 bootrom 的时钟配置中于整个 PVT 范围内有效。其他软件不得依赖这些缓解措施。
- **缓解 RP2350-E16:USB\_OTP\_VDD 中断可能导致 OTP 行读取数据损坏。** 下列更改适用：
  - 对 OTP PSM 及 OTP 读取电路进行了多项调整，以检测不可靠的操作。
  - RISC-V 调试现已由 CRIT1.SECURE\_DEBUG\_DISABLE 禁用，除 CRIT1.DEBUG\_DISABLE 之外。（在 A2 版本中，仅使用后者位。）
  - CRIT0.ARM\_DISABLE 不再禁用 Arm 处理器。
  - 同时对 CRIT0.ARM\_DISABLE 和 CRIT0.RISCV\_DISABLE 进行编程时，将被视为非法组合，设备将无法启动。

- 更新以下时钟配置寄存器的复位状态：
  - ROSC: FREQA.DS0\_RANDOM 和 FREQA.DS1\_RANDOM 由 0 调整为 1，启用前两驱动级的随机化。
  - CLOCKS: CLK\_SYS\_CTRL.SRC 从 0 变更为 1（选择 AUX 源）。
  - CLOCKS: CLK\_SYS\_CTRL.AUXSRC 从 0 变更为 2（选择 ROSC 作为 AUX 源）。

## Bootrom 变更

A3 只读存储器引导程序引入了以下更改：

- 修复 RP2350-E10：带有分区表的 UF2 拖放功能失效。此前需要在 picotool 中采用变通方法，但 A3 只读存储器引导程序已不再需要该变通方法。picotool 保留了该变通方法，以确保与 A2 的兼容性。
- 修复 RP2350-E13：包含显式无效 IMAGE\_DEF 后紧跟有效 IMAGE\_DEF（按此顺序）的二进制文件无法启动。
- 修复 RP2350-E14：只读存储器引导程序的 connect\_internal\_flash() 函数始终使用引脚 0，忽略任何已配置的 FLASH\_DEVINFO\_CS1 片选引脚。
- 修复 RP2350-E15：只读存储器引导程序的 otp\_access() 函数对第 62 和第 63 页应用了错误的访问权限。
- 修复 RP2350-E19：重启时若 FRCE\_OFF 中设置了特定位，RP2350 重启会停止。
- 缓解 RP2350-E20：攻击者若具备对芯片的物理访问权限且能在精准时机实施物理故障注入，可能通过针对合法非安全调用的 bootrom reboot() 函数，在受保护的 RP2350 上执行未签名代码。
- 缓解 RP2350-E21：攻击者若具备对芯片的物理访问权限且能在精准时机实施物理故障注入，可能从处于 BOOT\_SEL 模式的 RP2350 提取 OTP 中的敏感数据。
- 修复 RP2350-E22：解析格式错误的 lollipop 块循环时，系统应失败而非停止运行。
- 修复 RP2350-E23：PICOBLOCK 的 GET\_INFO 命令始终返回 PACKAGE\_SEL 为零。
- 缓解 RP2350-E26：RCP 随机延迟可能导致侧信道攻击。这些延迟已在 bootrom 中禁用。
- 更新早期引导路径，将 clk\_ref 分频器从 1 更改为 5，ROSC 分频器从 8 更改为 2。
  - 结合寄存器复位状态更改，此举将引导 clk\_sys 频率提高 4 倍，约为 48 MHz。
  - 这些更改缩短了引导时间，降低了故障注入的易感性。
  - 这些更改适用于所有引导结果，包括看门狗和 POWMAN 向量引导。

## RP2350 A4

步进版本 A4 通过 CHIP\_ID.REVISION 值为 0x8 进行识别。

## 硬件变更

该步进版本未涉及硬件更改。

## Bootrom 变更

A4 bootrom引入以下更改：

- 修正**RP2350-E18**：当FLASH\_PARTITION\_SLOT\_SIZE包含无效ECC位模式时，**RP2350**将永远无法引导。该问题源自**RP2350-E17**（对单个ECC OTP行的受保护读取操作若相邻行数据非有效ECC数据，则会引发故障）。基础硬件问题尚未解决，但bootrom在该情况下避免了此问题。
- 缓解**RP2350-E24**：攻击者若具备物理访问芯片的权限、中等硬件资源及能够在精准时机对CPU施加物理故障，可导致加固版RP2350执行未签名代码。A4只读存储器包含针对该漏洞及其它基于相同底层机制潜在漏洞的额外故障注入缓解措施。
- 修复**RP2350-E25**：此前使用非字长尺寸的LOAD\_MAP未触发错误。只读存储器现已正确拒绝这些结构。

# 附录E：勘误表

按章节字母顺序排列。

## 访问控制

### RP2350-E3

参考资料	RP2350-E3
摘要	在 QFN-60 封装中， <b>GPIO_NSmask</b> 控制了错误的 <b>PADS</b> 寄存器。
影响范围	仅限 RP2350 A2，QFN-60 封装
描述	<p>RP2350 对 IO 口、其控制寄存器及 ADC 通道进行了重映射，使两种封装尺寸的 GPIO 编号均连续，尽管因物理设计限制，QFN-60 封装仅引出稀疏选择的 IO 引脚。</p> <p>GPIO_NSmask0/GPIO_NSmask1 寄存器与 <b>PADS</b> 寄存器之间的映射未考虑该重映射。因此，仅在 QFN-60 封装中，<b>GPIO_NSmask0</b> 寄存器位被错误地应用于某些引脚的寄存器。具体而言，<b>PADS_BANK0</b> 寄存器 29 至 0 由 <b>GPIO_NSmask</b> 位 47 至 44、39 至 33、30 至 28、24 至 17 及 15 至 8 的级联组合控制（所有范围均含）。</p> <p>这意味着授予对 QFN-60 封装中 <b>PADS</b> 寄存器的非安全访问权限，不允许非安全软件控制正确的引脚。此外，可能会允许对 <b>GPIO_NSmask0</b> 中未授权引脚的非安全控制。</p> <p>QSPI <b>PADS</b> 寄存器（Bank 1）不受影响，因为它们未针对不同封装进行重映射。</p>
解决方案	<p>通过清除 <b>PADS_BANK0.NSP</b> 和 <b>NSU</b> 位，禁用对 <b>PADS</b> 寄存器的非安全访问。</p> <p>实施安全网关（Arm 架构）或 <b>ecall</b> handler（RISC-V 架构），以允许非安全/U 模式代码访问其分配的 <b>PADS_BANK0</b> 寄存器。</p>
修复者	RP2350 A3，文档，软件

## Bootrom

### RP2350-E10

参考资料	RP2350-E10
摘要	含分区表时 UF2 拖放无法正常工作
影响范围	RP2350 A2
描述	将 UF2 拖入 USB 大容量存储设备时，芯片版本为 A2 的引导只读存储器不会在检查分区表前设置闪存。若存在分区表，UF2 下载将失败。

解决方案	<p>在 UF2 起始处添加一个带有 Absolute 系列 ID 的单独块，目标为闪存末端，块编号为 0，块数为 2。该块首先写入闪存，但不会重启设备，同时为 UF2 后续数据的正确下载配置闪存。</p> <p>SDK 中的 <code>picotool</code> 会自动处理此项，若指定 <code>--abs-block</code> 标志，生成的 UF2 会包含该块。</p> <p>该解决方法意味着下载此类 UF2 时，闪存末尾的块将被擦除，可能覆盖用户数据。</p> <p>截至 <code>picotool</code> 版本 2.1.0，该额外的 UF2 块被标记为 Raspberry Pi 专用 UF2 扩展 <code>UF2_EXTENSION_RP2_IGNORE_BLOCK (0x9957e304)</code>。RP2350 A3 及以后版本的 bootrom 已包含该缺陷的修复，因此无需采用此变通方法。UF2 块中存在该扩展，使较新版本的 RP2350 能识别并忽略该变通块，从而避免覆盖用户数据的风险。</p>
修复者	RP2350 A3 bootrom, 文档, 软件

## RP2350-E13

参考资料	RP2350-E13
摘要	包含一个明确无效的 <code>IMAGE_DEF</code> 且紧跟一个有效的 <code>IMAGE_DEF</code> （二者按此顺序）的二进制文件将无法启动。
影响范围	RP2350 A2
描述	<p>当二进制文件的块循环中包含一个明确无效的 <code>IMAGE_DEF</code>，且该无效的 <code>IMAGE_DEF</code> 位于有效的 <code>IMAGE_DEF</code> 之前时，RP2350 将无法从该二进制文件启动。</p> <p>若满足以下任一条件，<code>IMAGE_DEF</code> 即被视为明确无效：</p> <ul style="list-style-type: none"> <li>该 <code>IMAGE_DEF</code> 用于 RP2040。</li> <li>该版本无回滚版本，并且在 OTP 中已设置 <code>BOOT_FLAGS0.ROLLBACK_REQUIRED</code> 标志</li> </ul>
解决方案	<p>请使用 <code>IGNORED</code> 项替代显式无效的 <code>IMAGE_DEF</code>。RP2040 启动二进制文件时无需 <code>IMAGE_DEF</code>，且在使用回滚机制时，无效的 <code>IMAGE_DEF</code> 将被忽略。</p> <p>SDK 对 RP2040 二进制文件采用此解决方案。设置 <code>PICO_CRT0_INCLUDE_PICOBIN_BLOCK</code> 后，SDK 对 RP2040 二进制文件使用 <code>IGNORED</code> 项替代 <code>IMAGE_DEF</code>。您可通过设置 <code>PICO_CRT0_INCLUDE_PICOBIN_IMAGE_TYPE_ITEM</code> 来覆盖此行为，使用 <code>IMAGE_DEF</code>。更多示例详见 <code>pico-examples</code> 中的通用二进制文件。</p> <p><code>picotool</code> 采用此解决方案处理回滚版本。使用 <code>picotool seal</code> 封装二进制文件并添加回滚版本时，会将首个无回滚版本的 <code>IMAGE_DEF</code> 转换为 <code>IGNORED</code> 项。</p>
修复者	RP2350 A3 bootrom, 文档, 软件

## RP2350-E14

参考资料	RP2350-E14
摘要	bootrom 的 <code>connect_internal_flash()</code> 函数始终使用引脚 0，忽略任何已配置的 <code>FLASH_DE_VINFO</code> 中 <code>CS1</code> 芯片选择引脚
影响范围	RP2350 A2

描述	<p>在使用 bootrom 函数 <code>connect_internal_flash()</code> 配置 CS1（例如在闪存启动过程中）时，bootrom 始终将引脚寄存器配置为引脚 0，忽略 <code>FLASH_DEVINFO</code> 中指定的任何 CS1 引脚。</p> <p>因此，指定的 CS1 引脚保持隔离状态（详见第 9.7 节）。除非将 CS1 连接至引脚 0，否则无法访问连接至第二芯片选择的 QSPI 设备。</p> <p><code>FLASH_DEVINFO</code> 可在 OTP 中配置，亦可在运行时配置。详情请参阅 <code>flash_devinfo16_ptr</code>。</p>
解决方案	使用 bootrom 函数 <code>connect_internal_flash()</code> 后，需手动配置 CS1 引脚寄存器以解除隔离状态。或者，将 CS1 连接至引脚 0。
修复者	RP2350 A3 bootrom, 文档, 软件

## RP2350-E15

参考资料	RP2350-E15
摘要	bootrom 的 <code>otp_access()</code> 函数对第 62 页和第 63 页应用了错误的访问权限。
影响范围	RP2350 A2
描述	<p>bootrom <code>otp_access()</code> 函数错误地将 OTP 行 PAGE62_LOCK1 和 PAGE63_LOCK1 中指定的访问权限应用于对应的整个 OTP 页（第 62 页和第 63 页）。此说法不正确，因为第 62 页和第 63 页包含针对其他页面的锁定字：每个锁定字实际上受相应页面权限保护。</p> <p>ATE 编程随后锁定非安全软件和引导程序对第 63 页锁定字的写入权限（以防止非安全地设置 RMA 标志），并锁定非安全软件对第 62 页锁定字的写入权限。这防止非安全软件修改任何 OTP 页面锁定，并阻止引导程序修改第 32 至 63 页的锁定。</p>
解决方案	<p>在设备上运行代码时，请勿使用非安全的 <code>otp_access()</code> 函数设置 OTP 页面锁定。若需从非安全代码设置 OTP 页面锁定，请实现您自己的安全 API，允许从非安全代码调用该 API 完成此操作。</p> <p>第 32 至 63 页 OTP 页面的页面锁定可通过 <code>picotool</code> 使用 <code>picotool otp permissions</code> 命令设置。该命令将一个安全二进制文件加载至设备上的 XIP SRAM，以更改权限，然后重新启动进入 USB 引导程序。</p>
修复者	RP2350 A3 bootrom, 文档, 软件

## RP2350-E18

参考资料	RP2350-E18
摘要	如果 <code>FLASH_PARTITION_SLOT_SIZE</code> 包含无效的 ECC 位模式，RP2350 将永远无法启动。
影响范围	RP2350 A2, RP2350 A3

描述	<p>如果ECC行编程被中断，ECC行可能包含无法通过ECC验证的值。由于任何ECC行都可能包含无效的、部分写入的值，启动只读存储器（bootrom）使用OTP中的单独“启用”标志来指示特定ECC行是否预期包含有效值。用户应仅在确认特定ECC行已正确写入后设置此标志。</p> <p>对于 <code>FLASH_PARTITION_SLOT_SIZE</code>, 该“启用”标志是 <code>BOOT_FLAGS0.OVERRIDE_FLASH_PARTITION_SLOT_SIZE</code> 来覆盖此值。</p> <p>在 <code>FLASH_PARTITION_SLOT_SIZE</code> 的情况下，启动只读存储器会先读取该行值并断言该值有效，然后再检查启用标志，因此如果OTP中 <code>BOOT_FLAGS0.OVERRIDE_FLASH_PARTITION_SLOT_SIZE</code> 行包含无效的EC值，启动过程将会挂起。</p>
解决方案	请勿编程 <code>FLASH_PARTITION_SLOT_SIZE</code> ，或者至少应明确注意，如果编程操作被中断或失败，可能会导致设备变砖。
修复者	RP2350 A4 只读存储器，文档

## RP2350-E19

参考资料	RP2350-E19
摘要	当重新启动时，如果 <code>FRCE_OFF</code> 中的某些位被设置，RP2350 重启将会卡死。
影响范围	RP2350 A2
描述	<p>启动路径中存在错误断言，假设除了 <code>FRCE_OFF.PROC1</code> 之外的所有位均为清零状态。这些位仅在用户设置后并重新进入启动路径时，才可能在启动期间被设置。</p>
解决方案	请勿在除 <code>FRCE_OFF.PROC1</code> 外的其他位被设置时执行 <code>WATCHDOG</code> 或 <code>POWMAN</code> 启动，或对 <code>core0</code> 进行复位。
修复者	RP2350 A3 只读存储器，文档

## RP2350-E20

参考资料	RP2350-E20
摘要	具有物理访问权限并能在精确时机对 CPU 施加物理“故障注入（glitch）”的攻击者，可能通过针对引导只读存储器中的合法非安全调用 <code>reboot()</code> 函数，使受保护的 RP2350 运行未经签名的代码。
影响范围	RP2350 A2
描述	<p>RP2350 引导只读存储器提供 <code>reboot()</code> 函数用于重新启动 RP2350。该方法可能通过 <code>PICOBOOT</code> 接口（例如 <code>picotool</code>）对非安全调用者开放，或在相应权限被授予时，对设备上的非安全代码开放。</p> <p>在从非安全代码调用 bootrom 的 <code>reboot()</code> 函数时，特定的重启类型（<code>REBOOT_TYPE_PC_SP</code>）不被允许，因为该类型会在重启后以安全状态启动用户提供的代码。<code>reboot()</code> 函数在从非安全上下文调用时，会正确禁止该重启类型。然而，如果传递给函数的是有效的重启类型（例如 <code>REBOOT_TYPE_NORMAL</code>），则一个时序精准的处理器故障可能导致错误的代码路径被执行，从而配置 <code>WATCHDOG</code> 暂存寄存器，允许在重启后安全地执行用户提供的代码。</p>

解决方案	<p>如果不需要任何基于 WATCHDOG 的非正常启动类型（包括程序化重启进入 BOOTSEL 模式及用于 A/B 分区的重要 FLASH_UPDATE 启动），则可以设置 OTP 标志 BOOT_FLAGS0.DISABLE_WATCHDOG_SCRATCH，使启动过程中完全忽略 WATCHDOG 暂存寄存器，从而通过 WATCHDOG 复位只能执行常规启动。</p> <p>一种更为细致的方法是禁止非安全代码调用 reboot() 函数。这是由安全应用启动的非安全代码的默认情况（见第 5.4.2 节）。然而，BOOTSEL 模式的引导程序本身是一个非安全应用，能够访问该函数。</p> <p>然而，BOOTSEL 模式可通过 BOOT_FLAGS0.DISABLE_BOOTSEL_UART_BOOT 禁用，若不需要时， BOOT_FLAGS0.DISABLE_BOOTSEL_USB_PICOBOOT_IFC, and BOOT_FLAGS0.DISABLE_BOOTSEL_USB_MSD_IFC。</p> <p>BOOT_FLAGS0.DISABLE_BOOTSEL_USB_PICOBOOT_IFC 是最关键的，因为 PICOBOOT 为用户向引导只读存储器的 reboot() 函数传递特定参数提供了通道。然而，BOOTSEL 模式的任何其他使用在未来针对非安全代码的某些攻击中可能存在安全漏洞。</p>
鸣谢	Marius Muench
修复者	RP2350 A3 只读存储器，文档

## RP2350-E21

参考资料	RP2350-E21
摘要	攻击者如能对芯片进行物理访问，并能在精确时刻对 CPU 实施物理“故障注入”，有可能从处于 BOOTSEL 模式的 RP2350 的一次性可编程只读存储器（OTP）中提取敏感数据。
影响范围	RP2350 A2
描述	<p>攻击者若具备对芯片的物理访问权限，能够在进入 BOOTSEL 模式时精准执行物理故障注入攻击，可能导致部分针对 BOOTSEL 模式的 OTP 页访问权限未被正确应用，从而存在敏感数据泄露的潜在风险。</p> <p>RP2350 的 BOOTSEL 模式通过 PICOBOOT 暴露了 API，用户可以通过 picotool 读取或写入 OTP 行。部分 OTP 行（例如加密密钥或其他机密信息）不应通过该方式被读取。同样，某些行可能受到写入保护。此项安全措施由 OTP 中存储的页锁（每页包含 64 行）进行管理。</p> <p>进入 BOOTSEL 模式时，应对 OTP 进行锁定，确保任何软件（包括安全软件）均无法访问未标记为 BOOTSEL 模式可访问的内容。然而，通过两次精确时序的处理器故障，可阻止页面被正确锁定。</p>
解决方案	<p>1. 通过 BOOT_FLAGS0.DISABLE_BOOTSEL_UART_BOOTT、BOOT_FLAGS0.DISABLE_BOOTSEL_USB_PICOBOOT_IFC 和 BOOT_FLAGS0.DISABLE_BOOTSEL_USB_MSD_IFC 完全禁用 BOOTSEL 模式的引导加载程序。</p> <p>BOOT_FLAGS0.DISABLE_BOOTSEL_USB_PICOBOOT_IFC 最为重要，因为 PICOBOOT 提供用户访问 OTP 的通道，但 BOOTSEL 模式的其他任何使用，在结合未来对非安全代码的攻击时，均可能存在漏洞。</p> <p>1. 使用 OTP 访问密钥（第 13.5.2 节）保护您不希望通过 BOOTSEL 模式访问的 OTP 数据，但这仅在数据仅于应用程序能够提供密钥时才有效。</p>
鸣谢	Thomas Roth

修复者	RP2350 A3 只读存储器, 文档
-----	---------------------

## RP2350-E22

参考资料	RP2350-E22
摘要	解析格式错误的 "lollipop" 块循环将导致系统挂起, 而非失败。
影响范围	RP2350 A2
描述	<p><code>PARTITION_TABLE</code>和<code>IMAGE_DEF</code>的元数据作为块循环的一部分存储。这些块循环在启动期间及其他时间被解析。</p> <ul style="list-style-type: none"> <li>“棒棒糖”（lollipop）块循环是一种无效块循环，指从最后一个块回环至非第一个块的循环。此类无效块循环绝不会由SDK或picotool生成；但可能会由其他工具生成。</li> </ul>
解决方案	请勿使用“棒棒糖”块循环。若将“棒棒糖”块循环编程进闪存以供启动时读取, 将导致启动挂起, 并且进入BOOTSEL模式时同样挂起。因此, 为重新启用启动, 须通过其他方式清除闪存, 例如通过调试器利用SWD接口。
修复者	RP2350 A3 只读存储器, 文档

## RP2350-E23

参考资料	RP2350-E23
摘要	PICOBOOT GET_INFO命令对PACKAGE_SEL始终返回零。
影响范围	RP2350 A2
描述	<p>PICOBOOT GET_INFO命令可用于获取系统信息, 其方式类似于bootrom中的<code>get_sys_info()</code>函数。该信息包括PACKAGE_SEL读取值, 用以指示RP2350封装类型为QFN60还是QFN80。</p> <p>进入 BOOTSEL 模式时, <code>SYSINFO</code> 块错误地保持复位状态, 因此通过 PICOBOOT 访问该寄存器时, 读取值为零。</p> <p>该问题不影响从用户代码中调用 bootrom 的 <code>get_sys_info()</code> 函数本身的使用。</p>
解决方案	应通过 PICOBOOT 的 OTP_READ 命令读取 <code>register_link_macro:[register=OTP_DATA_NUM_GPIOS_ROW]</code> 寄存器, 以确定封装尺寸。此变通方法由 picotool 采用。
修复者	RP2350 A3 bootrom, 文档, 软件

## RP2350-E24

参考资料	RP2350-E24
摘要	具备物理访问芯片权限、中等硬件设备以及能够在精确时刻对 CPU 实施物理“干扰”攻击的攻击者, 可能在受保护的 RP2350 上执行未经签名的代码。
影响范围	RP2350 A2, RP2350 A3

描述	具备物理访问芯片权限并能够在启动过程中通过 QSPI 在准确时刻切换 RP2350 读取的“闪存”内容的攻击者，结合对 CPU 精确时机的物理“干扰”攻击，可能诱使 bootrom 在安全启动期间检查内存中加载的非程序二进制数据的签名。  如果该“其他”数据通过签名校验，则攻击者的二进制文件将在未通过签名校验的情况下被执行，这使得用户能够在 RP2350 上运行任意未经签名的代码。
鸣谢	Kévin Courdesses (参见 <a href="https://courk.cc/rp2350-challenge-laser#flash-memory-organization">https://courk.cc/rp2350-challenge-laser#flash-memory-organization</a> )
解决方案	无
修复者	RP2350 A4 只读存储器启动程序

## RP2350-E25

参考资料	RP2350-E25
摘要	使用非字长大小的 LOAD_MAP 不会引发错误。
影响范围	RP2350 A2, RP2350 A3
描述	在 <code>LOAD_MAP</code> 中，不支持非字长大小，且文档对此已有明确说明。然而，目前这种情况尚不会引发错误。尽管非字长大小在某些特定情况下目前或许可用，但您绝不应使用它们，因为它们可能不会在所有情况下如预期般工作，并且未来可能被正式视为错误。  SDK 与 <code>picotool</code> 不会生成包含非字长大小条目的 <code>LOAD_MAP</code> 。
解决方案	请勿在 <code>LOAD_MAP</code> 中使用非字长（非4字节的整数倍）大小。最佳实践是确保链接器的内存段既为字长大小且字长对齐。  自 RP2350 A4 只读存储器引导程序版本起，在检查 <code>LOAD_MAP</code> 时若检测到非字长大小，
修复者	RP2350 A4 只读存储器，文档

## 总线结构

### RP2350-E27

参考资料	RP2350-E27
摘要	APB 和 FASTPERI 仲裁器的总线优先级控制适用于错误的管理者。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4

描述	<p>RP2350总线结构主要由一个AHB5交叉开关组成，其中6个上游端口（管理者）被路由至15个下游交叉开关端口。图5展示了包括该交叉开关在内的总线结构整体架构。由于任一时钟周期内可对给定下游交叉开关端口发起多次访问，仲裁电路选择一个传输转发至该端口，并阻塞所有其他针对该端口的传输。</p> <p>BUSCTRL BUS_PRIORITY寄存器用于控制这些仲裁电路。该寄存器为以下四组AHB5管理者中的每一组配置1位优先级等级：</p> <ul style="list-style-type: none"> <li>• DMA写入</li> <li>• DMA读取</li> <li>• 核心0指令获取及核心0加载/存储</li> <li>• 核心1指令获取及核心1加载/存储</li> </ul> <p>来自高优先级管理器的访问始终优先于来自低优先级管理器的访问。同一优先级管理器的多个访问依次单个处理，按照循环轮询的方式进行仲裁。</p> <p>在FASTPERI和APB仲裁器中，这些信号接线错误，导致优先级判定错误，错误的管理器被优先处理：</p> <ul style="list-style-type: none"> <li>• <a href="#">BUS_PRIORITY.PROC0</a> 控制 DMA 写入的优先级</li> <li>• <a href="#">BUS_PRIORITY.PROC1</a> 控制核心0加载/存储的优先级</li> <li>• <a href="#">BUS_PRIORITY.DMA_R</a> 控制核心1加载/存储的优先级</li> <li>• <a href="#">BUS_PRIORITY.DMA_W</a> 控制 DMA 读取的优先级</li> </ul> <p>BUS_PRIORITY 控制正确应用于所有其他仲裁器：只读存储器、SRAM 和 XIP。</p> <p>例如，如果设置了 <a href="#">DMA_R</a> 和 <a href="#">DMA_W</a> 位，将使 DMA 访问优先于处理器访问只读存储器、SRAM 和 XIP，但外围设备访问时，DMA读取和核心1加载/存储优先于 DMA写入和核心0加载/存储。</p>
解决方案	<p>目前尚无完整的修复方案，但通常仍可通过为外设配置正确优先级的<a href="#">BUS_PRIORITY</a>实现必要的优先级排序，然后在<a href="#">SRAM</a>中安排缓冲区以最小化争用，例如使用<a href="#">SRAM8</a>或<a href="#">SRAM9</a>作为处理器专用内存。</p> <p>还应考虑以下方法：</p> <ul style="list-style-type: none"> <li>• 将RAM访问分散到SRAM0至SRAM3与SRAM4至SRAM7的条带区域，以进一步减少RAM争用。</li> <li>• 尝试通过对支持的外设使用更宽的数据访问，降低总体外设带宽需求。例如，SPI支持16位数据，HSTX和PIO支持32位数据。</li> <li>• 避免处理器轮询外设状态寄存器。应改为使用中断或DMA DREQ信号。</li> <li>• 评估默认的轮询仲裁是否优于可达的非对称优先级配置。</li> </ul>
修复者	文档

## DMA

### RP2350-E5

参考资料	RP2350-E5
摘要	活动通道中 <b>CHAIN_TO</b> 与 <b>ABORT</b> 的交互
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>CHAN_ABORT 寄存器指令 DMA 通道停止发起传输，并在当前传输完成后清除其 <b>BUSY</b> 标志。此功能最初用于恢复那些因 DREQ 持续低电平而卡住的通道。通过向 CHAN_ABORT 写入一个中止通道的位图来启动 <b>ABORT</b> 操作。</p> <p>各位保持置位状态，直到对应通道停止工作。</p> <p>该勘误包含两种行为：首先，终止某通道会触发其 <b>CHAIN_TO</b> 信号，前提是被终止的通道为最后一个完成写传输的通道。其次，处于 <b>ABORT</b> 状态的通道可能在 <b>ABORT</b> 寄存器清零的最后一个周期之前被重新触发，因为该通道在该周期既非活动状态且被启用，且 <b>ABORT</b> 本身不阻止触发。但由于 <b>ABORT</b> 仍处于生效状态，传输计数保持为零。在 <b>ABORT</b> 结束的下一个周期，该通道因传输计数器为零而完成操作。这导致该通道的 IRQ 及 <b>CHAIN_IN_TO</b> 信号在 <b>ABORT</b> 完成后的周期被触发。</p> <p>当中止相互链式连接的多个通道时，这两种行为存在问题，因为它们可能导致通道在中止后立即重新启动。</p>
解决方案	在中止一个活动通道之前，请清除被中止通道及其链式连接的所有通道的 <b>EN</b> 位（CH0_CTRL_TRIG.EN）。此操作确保通道不会受到重新触发的影响。
修复者	文档

## RP2350-E8

参考资料	RP2350-E8
摘要	对于长度为零的传输， <b>CHAIN_TO</b> 可能不会触发。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p><b>CTRL.CHAIN_TO</b> 字段配置通道在完成其预设传输序列后启动另一个通道。<b>CHAIN_TO</b> 在通道最后一次写入完成的周期内发生，而被链式启动的通道将在下一个周期激活。</p> <p>硬件实现假设 <b>CHAIN_TO</b> 始终发生于写入完成之后。</p> <p>当通道以传输计数为零触发时，情况并非如此；此时通道会在触发后的下一个周期完成，而不执行任何总线访问。</p> <p>仅当以传输计数为零启动的通道是最后完成写入传输的通道时，才会触发其 <b>CHAIN_TO</b>。仅当所述通道此前已执行非零长度的传输序列，且此后未有其他通道完成写操作时，方成立。</p>
解决方案	请勿将 <b>CHAIN_TO</b> 与零长度传输同时使用。应避免在控制块列表中间出现零长度传输，若可能，应以虚拟传输替代。
修复者	文档

## GPIO

## RP2350-E9

参考资料	RP2350-E9
摘要	当 Bank 0 GPIO 使能输入时，漏电流增加。
影响范围	RP2350 A2

<p><b>描述</b></p> <p>针对 GPIO 引脚 0 至 47：</p> <p>当 Bank 0 GPIO 引脚配置为输入且引脚电压位于 <math>V_{IL}</math> 与 <math>V_{IH}</math> 之间的未定义逻辑区间时，漏电流增加。</p> <p>当引脚设为输入状态（启用输入使能且禁用输出使能）且引脚电压处于未定义逻辑区间内时，漏电流将超过标准规定的 <math>I_{IN}</math> 漏电电平。在该状态下，引脚可能输出电流（具体数值取决于芯片及引脚电压，通常约为 <math>120\mu A</math>）。该漏电流会使引脚电压维持在约 <math>2.2 V</math>，因该电压为漏电流的有效源电压，仅能由适当低阻抗驱动器或上拉电阻克服。</p> <p>如果启用，垫脚下拉电阻在此状态下明显弱于泄漏电流，因此其强度不足以将垫脚电压拉低。</p> <p>使用低阻抗 (<math>8.2 k\Omega</math> 或以下) 电源驱动或拉低垫脚输入，将克服错误泄漏电流，使电压降至泄漏电流发生阈值以下，因此在此情况下，若垫脚被驱动或拉低，将保持低电平。</p> <p>错误泄漏仅在垫脚输入使能开启时发生且持续存在；禁用输入使能将重置（消除）该泄漏。</p> <p>垫脚上拉仍然有效。若启用，它会将垫脚拉至 <math>IOVDD</math> 电位，从而将输入电压拉出问题区间。</p> <p>上述电压和电流均基于 <math>3.3 V</math> <math>IOVDD</math>。对于 <math>1.8 V</math> <math>IOVDD</math>，泄漏的有效源电压为 <math>1.8 V</math>，峰值电流约为 <math>30\mu A</math>。当引脚电压处于 <math>0.6 V</math> 至 <math>1.8 V</math> 之间时，此状态实际上构成一个上拉（与标准引脚上拉独立）。</p> <p>这些图表展示了典型芯片的泄漏电流与引脚输入电压之间的关系，<math>IOVDD</math> 为 <math>3.3 V</math>。  <a href="#">图153及1.8 V 图154</a>。</p> <p>具体而言，该问题在以下条件下发生，适用于任意 GPIO 0 至 47：</p> <ol style="list-style-type: none"> <li>1. 引脚电压处于未定义逻辑区间。</li> <li>2. <math>GPIO0.IE</math> 中启用了输入缓冲。</li> <li>3. 输出缓冲被禁用（例如选择了 <math>NULL</math> GPIO 功能）。</li> <li>4. <math>GPIO0.ISO</math> 中的隔离被解除，或在设置隔离时上述条件成立。</li> </ol> <p>当所有上述条件均满足时，引脚输入泄漏可能超过规格限值。</p> <p>该问题可能影响多种常见电路：</p> <ul style="list-style-type: none"> <li>• 依赖浮动引脚保持低泄漏电流的电路。</li> <li>• 依赖内部下拉电阻的电路。</li> </ul> <p>如果内部上拉被使能，任何悬空信号将被拉高，从而消除因多余漏电仅作为电流源而造成的泄漏增加。当然，如引脚由强源供电，例如强势分压器，则无法防止泄漏增加。</p> <p>由于输入使能位初始为清除状态，此情况不会影响 PoR 或 RUN 复位后紧随其后的引脚下拉行为。在该状态下，下拉电阻功能保持正常。</p> <p>该问题不影响使用不同引脚宏且无故障电路的 QSPI 引脚。USB PHY 的引脚同样不受影响。</p> <p>该问题确实影响采用与 Bank 0 GPIO 相同容错引脚宏的 SWD 引脚，但由于 SWD 引脚默认上拉，因此无不良影响。</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

解决方案	<p>若需引脚下拉行为，请清除 GPIO0.IE 中针对 GPIO0 至 GPIO47 的引脚输入使能位，以确保引脚下拉电阻将引脚信号拉低。软件读取下拉 GPIO 引脚状态时，应在读取前立即设置 GPIO0.IE 以启用输入缓冲器，读取后立即禁用。如果引脚已处于逻辑0状态，重新启用输入不会影响下拉状态。</p> <p>亦可使用8.2 kΩ或更小的外部下拉电阻。</p> <p>PIO程序无法切换引脚控制，因此视具体应用可能需要外部下拉电阻。</p> <p>按常规操作，若某引脚正在使用ADC通道，应按照第12.4.3节清除相关GPIO输入使能。</p>
修复者	RP2350 A3, 文档

图153。GPIO引脚泄漏电流，  
IOVDD=3.3 V

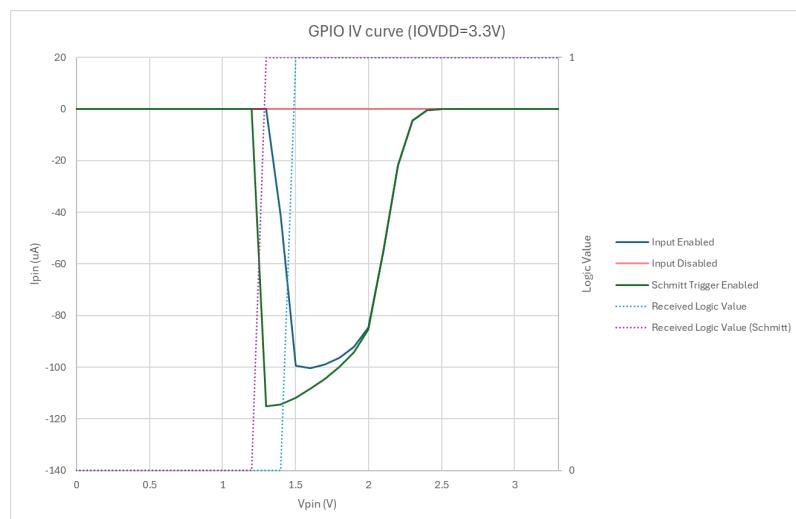
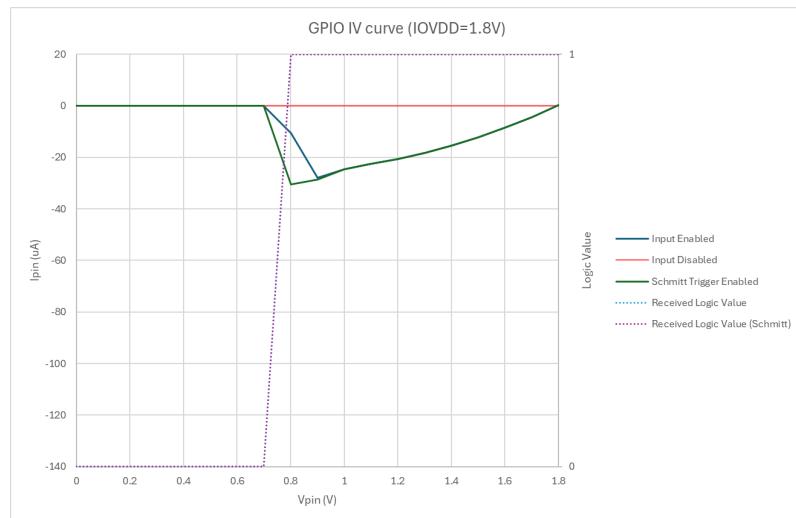


图154。GPIO引脚泄漏电流，  
IOVDD=1.8 V



## Hazard3

### RP2350-E4

参考资料	RP2350-E4
------	-----------

摘要	当核心1处于时钟门控睡眠状态时，系统总线访问（SBA）会无限期阻塞
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>系统总线访问（SBA）是RISC-V调试功能，允许调试模块在不依赖系统中计算核（hart）状态的情况下，直接访问系统总线。RP2350通过与核心1的加载/存储端口进行仲裁，实现调试模块对总线的访问以支持SBA功能。</p> <p>Hazard3 实现了由 MSLEEP CSR控制的定制低功耗状态。当设置MSLEEP.DEEPSLEEP时，Hazard3完全关闭其时钟，除唤醒所需的最小逻辑外。由于设计疏忽，这也导致SBA与负载/存储总线访问间的仲裁器被时钟门控。（此问题已在上游提交c11581e中修正。）</p> <p>因此，如果在核心1的MSLEEP.DEEPSLEEP被设置时启动SBA传输，且核心1处于等效WFI睡眠状态，则该SBA传输直到核心1从WFI状态唤醒前均不会取得任何进展。当启用的中断被断言或调试停止请求触发时，处理器将被唤醒。</p>
解决方案	<p>请配置调试转换器以避免使用SBA，或避免核心1进入时钟门控睡眠状态。A2启动只读存储器通过在核心1初始等待启动代码中不设置DEEPSLEEP以缓解该问题。</p> <p>处理器采用分层时钟门控综合设计，因此由DEEPSLEEP标志控制的顶层时钟门控相比默认WFI睡眠状态所带来的功耗节省极小。</p>
修复者	文档

## RP2350-E6

参考资料	RP2350-E6
摘要	PMPCFGx寄存器中的RWX字段顺序被转置。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>物理内存保护单元（PMP）定义了物理内存可配置范围的读、写及执行权限（RWX）。四个区域的RWX权限被打包到每个32位PMPCFG寄存器中，即PMPCFG0至PMPCFG3。</p> <p>依据RISC-V特权ISA规范，权限字段自MSB至LSB的排列顺序为X、W、R。</p> <p>Hazard3的实现顺序为R、W、X，导致使用正确位顺序的软件会误将读权限当作执行权限，反之亦然。（详见上游提交7d37029。）</p>
解决方案	配置PMP时，若X ≠ R，应采用该版本Hazard3实现的位顺序。在SDK中， <a href="#">hardware/regs/rvcsr.h</a> 寄存器头文件为RP2350构建提供了按实际实现顺序定义的位域。
修复者	文档

## RP2350-E7

参考资料	RP2350-E7
摘要	U模式不会忽略mstatus mie
影响范围	RP2350 A2、RP2350 A3、RP2350 A4

描述	MSTATUS.MIE位是面向M模式中断的全局使能位。软件通常会暂时清除此位，以确保关键短节在中断处理程序方面保持原子性。  RISC-V特权ISA规范要求，当hart处于较低特权模式时，特定特权模式的中断使能标志应被视为 1。在此情况下，内核处于U模式时，应将mstatus mie视为 1。  Hazard3未实现该规则，因此，在M模式中断被禁用的情况下进入U模式，将不会响应M模式中断。（参见上游提交a84742a。）
解决方案	通过 mret从M模式返回至U模式且mstatus.mpp == 0时，须确保mstatus.mpie被设置，以确保返回时IRQ被启用。
修复者	文档

## OTP

### RP2350-E16

参考资料	RP2350-E16
摘要	USB_OTP_VDD 的中断可能导致OTP行读取数据损坏。
影响范围	RP2350 A2
描述	<p>OTP阵列的读取电压由USB_OTP_VDD通过内部线性稳压器生成。虽然调节器具有“电源良好”信号，但该信号仅在初始上电复位启动序列期间被采样。对USB_OTP_VDD的外部干预可能导致阵列读取阶段锁存不正确的数据。</p> <p>错误行为包括但不限于：</p> <ul style="list-style-type: none"> <li>• 锁存阵列的前一次读取周期数据</li> <li>• 一个或多个位线对已编程位返回零</li> <li>• 字节移位的读取数据</li> </ul> <p>在受保护读取情况下，第一种故障模式可能导致保护读取检查通过，且保护字也被作为读取数据。若关键数据为启动时由OTP PSM采样的 CRIT0/CRIT1标志，则可能触发Hazard3调试并禁用Arm内核，导致CRIT1.S ECURE_BOOT_ENABLE与CRIT1.DEBUG_DISABLE标志的效果被撤销。</p> <p>受保护的ECC读取通常不易受到此类破坏，因为保护字是无效的ECC字，位删除或字节移位会可靠地使EC校验失败。</p> <p>RP2350 A3集成了更多防范错误 OTP 行为的保护措施。若以下任一检查未通过，芯片将被复位至 OTP PS M 阶段起始位置。</p> <ul style="list-style-type: none"> <li>• 在执行 OTP PSM 或用户访问操作时，OTP 稳压器 OK 信号会持续被监测。</li> <li>• 行读地址的第0位比特用于选择任意受保护读取时第一个或第二个只读存储器校准字（0x333333或0xffffffff），并据此进行验证。</li> <li>• 在 OTP PSM 中，CRIT0/1 行的保留-0 比特必须读取为0。</li> </ul> <p>无论芯片处于何种安全状态，均执行这些检查。可通过 AUXCTRL 寄存器的第2位比特对 OTP 稳压器检查进行掩码。</p>
鸣谢	Aedan Cullen（详见 <a href="https://github.com/aedancullen/hacking-the-rp2350">https://github.com/aedancullen/hacking-the-rp2350</a> ）

解决方案	无
修复者	RP2350 A3

## RP2350-E17

参考资料	RP2350-E17
摘要	对单个 ECC OTP 行执行受保护读取时，若相邻行中的数据非有效 ECC 数据，则会触发故障。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>OTP 中的每个“ECC 行”存储 16 位用户数据及用于纠正和/或检测位错误的纠错信息。ECC 行用于存储数据值，数据以完整的 16 位一次性写入 OTP。</p> <p>“受保护”的 ECC 行读取设计供 RP2350 启动路径或其他安全软件使用，当其期望读取 ECC 行且该行值无效时无法继续操作。通过“受保护”读取无效 ECC 值会导致芯片停机，直至重新启动。</p> <p>若 ECC 行编程被中断，ECC 行可能包含无法通过 ECC 校验的值。 由于任何 ECC 行都可能包含无效的、部分写入的值，启动只读存储器（bootrom）使用 OTP 中的单独“启用”标志来指示特定 ECC 行是否预期包含有效值。用户应仅在确认特定 ECC 行已正确写入后设置此标志。</p> <p>RP2350 OTP 硬件执行 ECC 读取时，实际上会读取一对行（从偶数行开始），但仅返回一行值。执行受保护 ECC 读取时，硬件会验证该对行的有效性，如任一行无效，受保护读取将导致停机。</p>
解决方案	<ul style="list-style-type: none"> <li>切勿将 ECC 行与 RAW 行存储于同一对行（以偶数行号开始的一对行）中，因 RAW 行通常不含有效的 ECC 值。但须注意，RAW 行中的零值为有效的 ECC 值。</li> <li>若受不同“enable”标志保护，切勿将两条 ECC 行存储于同一对行中。</li> </ul> <p>此解决方法适用于用户使用 OTP，但启动只读存储器（bootrom）预先存在的某些 ECC 行对违反了解决方案 2：</p> <ul style="list-style-type: none"> <li><a href="#">FLASH_DEVINFO</a> 与 <a href="#">FLASH_PARTITION_SLOT_SIZE</a></li> <li><a href="#">BOOTSEL_LED_CFG</a> 与 <a href="#">BOOTSEL_PLL_CFG</a></li> </ul> <p>为绝对安全，设置对中一半的“enable”标志后，不要更新并设置另一半的“enable”标志。若需安全设置两个 ECC 值，应先同时设置这两个值，再同时设置对应的两个“enable”标志。</p>
修复者	文档

## RP2350-E28

参考资料	RP2350-E28
摘要	第 62 及 63 页的 OTP 密钥适用于所有锁定字 0 至 63。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4

描述	<p>如13.5节所述，OTP最上方64个字（128行）包含每个128字节OTP页的保护信息。OTP的ECC数据总容量为<math>64 \times 128 \text{ B} = 8192 \text{ B}</math>，故每页对应一个锁定字。每个锁定字中的权限 <math>n</math> 覆盖 OTP 行 <math>64 * n</math> 至 <math>64 * n + 63</math>（含），且也涵盖锁定字本身。</p> <p>这使得锁定字62和63具有特殊性，因为它们没有任何关联的OTP页。这是因为这些页会与锁定字存储的位置发生重叠。因此，锁定字62和63应仅保护它们自身。该规则已正确应用于 <code>LOCK_NS</code> 和 <code>LOCK_S</code> 位的影响。然而，针对 <code>KEY_R</code>、<code>KEY_W</code> 和 <code>NO_KEY_STATE</code> 位的保护检查未能正确处理第62和63页。相反，它们仅通过将行号除以64来查找对应的锁定字。</p> <p>其结果是，锁定字0至31的密钥保护状态由 <code>PAGE62_LOCK0</code> 定义，且锁定字32至63的密钥保护状态由 <code>PAGE63_LOCK0</code> 定义。</p> <p>相反，锁字 0 至 61 的密钥配置不会影响这些锁字的可访问性。其仅影响由这些锁字保护的实际数据页的可访问性。</p>
解决方案	<p>作为部分缓解措施，出厂编程会撤销所有设备上第 62 和 63 页的非安全写入权限。此举防止非安全软件通过故意安装无效密钥来禁用对锁字的安全访问。有关空白设备上预先编程权限的完整列表，请参见第 13.5.5 节。该缓解措施适用于所有 RP2350 版本。</p> <p>软件不应依赖 OTP 访问密钥来保护锁字。</p>
修复者	文档

## RCP

### RP2350-E26

参考资料	RP2350-E26
摘要	RCP 随机延迟可能构成侧信道攻击。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>RCP 延迟通过协处理器暂停实现；这导致相关内核被完全暂停。由于内核在延迟期间被有效停止，若无其他总线管理器（如其他 CPU 或 DMA）活动，将显著减少芯片内门电路的切换活动。切换活动的减少引起 DVDD 电流降低，且延迟的典型持续时间使该降低在芯片外部可被检测。电流的降低及随后增加可能导致故障注入触发点。紧接RCP延迟操作之后的指令更容易被可靠定位，从而抵消时钟随机化的累积效应。</p> <p>RCP延迟概率分布的二阶效应表明，对于较大 <math>N</math>，执行 <math>N</math>条RCP指令后，增加的延迟趋于以 <math>N * 63</math>周期为中心的正态分布。因此，在已知一定数量RCP延迟之后的指令，统计上更容易被定位。</p> <p>基于上述两个因素，程序员应在安全代码中谨慎使用RCP延迟。特别应避免在以下情况下使用RCP延迟：</p> <ul style="list-style-type: none"> <li>• 可能多次执行的内层循环中。</li> <li>• 函数序言或尾声中的模板汇编代码部分。</li> <li>• 特别关键的操作前，例如修改 <code>ACCESSCTRL</code> 寄存器之前。</li> </ul> <p>作为缓解措施，自RP2350 A3版本起，bootrom对所有RCP指令均采用非延迟变体。</p>
解决方案	建议使用非延迟的RCP指令变体。

修复者	文档, 软件
-----	--------

## SIO

### RP2350-E1

参考资料	RP2350-E1
摘要	Interpolator OVERF 位因新的右旋转行为而损坏
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>RP2350 用右旋转替代了插值器的右移操作，从而能够合成左移操作。这在紧凑的地址生成循环中用于缩放索引寻址非常有用。</p> <p>OVERF 标志通过检查经过由 <code>CTRL_LANE0_MASK_MSB</code> 和 <code>CTRL_LANE1_MASK_MSB</code> 寄存器字段配置的 MSB 掩码屏蔽后仍为非零的后移位值中的位来工作。该标志用于丢弃超出由 <code>ACCU0</code> 和 <code>ACCU1</code> 表示的 [0, 1) UV 坐标环绕域之外的采样点，例如在仿射变换的精灵采样中。</p> <p>该问题产生的原因是右旋转会将非零的最低有效位旋转至最高有效位，这些非零位误触发了 OVERF 标志。</p>
解决方案	可通过检查 ACCU0 / ACCU1 的最高有效位手动计算 OVERF，或预先计算边界以避免对每个采样点进行检查。
修复者	文档

### RP2350-E2

参考资料	RP2350-E2
摘要	SIO SPINLOCK 写操作在 +0x80 偏移处被镜像
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>SIO 包含自旋锁寄存器，编号从 SPINLOCK0 至 SPINLOCK31。读取自旋锁寄存器时尝试获取锁，成功返回非零值，失败返回 0。写入自旋锁寄存器以释放该锁，因此下一次获取必定成功。SIO 自旋锁寄存器位于 SIO 寄存器偏移地址 <code>0x100</code> 至 <code>0x17c</code> 范围内。</p> <p>RP2350 在寄存器偏移地址 <code>0x180</code> 及以上新增了 SIO 寄存器：响铃门控（Doorbells）、PERI_NONSEC 寄存器、RISC-V 软中断寄存器、RISC-V MTIME 寄存器及 TMDS 编码器。</p> <p>SIO 地址解码器通过地址的第 8 位对自旋锁写操作进行检测。这意味着偏移地址范围 <code>0x180</code> 至 <code>0x1fc</code> 的写操作会被错误检测为对应偏移地址低 128 字节范围 <code>0x100</code> 至 <code>0x17c</code> 的自旋锁写操作。对任何这些高地址寄存器的写入将使相应的锁定状态变为未声明状态。</p> <p>此勘误仅影响对自旋锁寄存器的写入操作。读取操作得到正确解码，因此不受高于 <code>0x17c</code> 地址访问的影响。</p>

解决方案	<p>请使用处理器原子指令，替代SIO自旋锁。SDK的hardware_sync_spin_lock库在为RP2350构建时，默认使用软件锁变量，取代硬件自旋锁。</p> <p>以下SIO自旋锁可正常使用，因其不与任何可写寄存器地址重叠：5、6、7、10、11及18至31号。根据所使用的高地址SIO寄存器，部分其他锁地址亦可安全使用。</p> <p>18至24号锁与若干只读TMDS编码器寄存器地址重叠，该情况安全，因仅写入操作会被错误解码。</p>
修复者	文档，软件

## XIP

### RP2350-E11

参考资料	RP2350-E11
摘要	通过集合/方式操作清理XIP缓存时，会修改脏行的标记。
影响范围	RP2350 A2、RP2350 A3、RP2350 A4
描述	<p>0x1通过 set/way 清除缓存维护操作执行以下步骤：</p> <ol style="list-style-type: none"> <li>选择一个缓存行：地址位 12:3用来索引缓存组，位 13用于从构成每组两条8字节缓存行的组相中选择。</li> <li>检查该行是否包含未提交的写入数据（即 <b>dirty</b> 行）。</li> <li>如果该行为 dirty，则将数据写入下游并将该行标记为 <b>clean</b>。</li> </ol> <p>在第三步中，除了将该行标记为 clean 外，缓存控制器错误地将缓存行的标签设为发起该清理操作的维护写入地址位 25:13。缓存使用标签来识别当前驻留在每条缓存行中的众多可能的下游地址中的哪一个。</p> <p>因此，读取新标记的地址会返回来自原始地址的缓存数据，违反了内存契约。</p> <p>请考虑以下示例场景：</p> <ul style="list-style-type: none"> <li>QMI 窗口 0（起始地址为 <b>0x10000000</b>）连接有闪存设备</li> <li>QMI 窗口 1（起始于 <b>0x11000000</b>）连接有一台 PSRAM 设备。</li> <li>缓存中地址 <b>0x11000000</b>处于脏状态，且位于缓存的方式 0。</li> </ul> <p>程序员通过向地址 <b>0x18000001</b>写入数据开始清理缓存，清理的是集合 0 和方式 0。此操作清理了包含地址 <b>0x11000000</b>的脏缓存行。清理完成后，缓存将该行的标签更新为全零（对应维护写入的偏移地址）。随后，从 <b>0x10000000</b>的读取引发了伪缓存命中，返回了 PSRAM 数据，替代了闪存数据。</p> <p>有关缓存维护操作的详细内容，请参见第 4.4.1.1 节。有关缓存行状态及其转换的详细内容，请参见第 4.4.1.2 节。</p> <p>标签更新仅影响 <b>0x1</b>按集合/方式进行的清理；对于另外四种缓存维护操作，该方法要么正确，要么无害。</p>

解决方案	<p>为避免伪造缓存命中，请选择一个不会与来自QMI的缓存数据发生别名冲突的地址。该方法在清理脏行后，将其重新映射到QMI地址空间之外，因而导致下一次访问该脏地址时发生缓存未命中。SDK中的<code>xip_clean_all()</code>函数实现了该规避方案。</p> <p>更新后的标签是可预测的：始终为维护写操作的地址。例如，使用维护空间的高16 kB来清理所有缓存行：</p> <pre> 1 volatile uint8_t *maintenance_ptr = (volatile uint8_t*)0x1bffc001u; 2 for (int i = 0; i &lt; 0x4000; i += 8) { 3     maintenance_ptr[i] = 0; 4 }</pre> <p>由于对于无效、已清理或固定缓存行，清理操作无效，该规避方案不会干扰用于缓存作为SRAM的固定缓存行。</p>
修复者	文档，软件

## USB

### RP2350-E12

参考资料	RP2350-E12
摘要	USB状态信号同步不足
影响范围	RP2350 A2, RP2350 A3, RP2350 A4 (A3中缓解)

描述	<p>在USB外设中，某些主机端和设备端控制器事件从 <code>clk_usb</code>跨越至 <code>clk_sys</code>。此类信号多数缺乏适当的同步措施，无法保证在 <code>clk_sys</code>等于或低于 <code>clk_usb</code>时被正确采样。</p> <p>以下信号缺乏适当的同步措施：</p> <p><b>SIE_STATUS:</b></p> <ul style="list-style-type: none"> <li>• <code>TRANS_COMPLETE</code></li> <li>• <code>SETUP_REC</code></li> <li>• <code>STALL_REC</code></li> <li>• <code>NAK_REC</code></li> <li>• <code>RX_SHORT_PACKET</code></li> <li>• <code>ACK_REQ</code></li> <li>• <code>DATA_SEQ_ERROR</code></li> <li>• <code>RX_OVERFLOW</code></li> </ul> <p><b>INTR:</b></p> <ul style="list-style-type: none"> <li>• <code>HOST_SOF</code></li> <li>• <code>ERROR_CRC</code></li> <li>• <code>ERROR_BIT_STUFF</code></li> <li>• <code>ERROR_RX_OVERFLOW</code></li> <li>• <code>ERROR_RX_TIMEOUT</code></li> <li>• <code>ERROR_DATA_SEQ</code></li> </ul> <p>引导只读存储器的USB引导加载程序从 <code>pll_usb</code>生成 <code>clk_sys</code>。因此，两者时钟频率相同，且具有固定相位关系。在该条件及PVT极端情况下，实验室测试显示这些事件可能丢失，导致USB引导加载程序行为不稳定。</p> <p>RP2350 A3包含硬件修复，提升了对引导只读存储器关键时序信号的裕度，确保整个PVT范围内的可靠操作。但软件不得依赖该修复，故未作详细说明。</p>
解决方案	当外设处于使用状态时， <code>clk_sys</code> 的运行速度至少比 <code>clk_usb</code> 快10%。复位、挂起和恢复等准静态总线状态的信号不受该缺陷影响，因此在这些情况下， <code>clk_sys</code> 可以较低。
修复者	文档，软件

# 附录H：文档发布历史

## 2025年7月29日

- 新增硬件版本历史（附录C），并记录了A3和A4步进版本。
- 新增错误修订。
- 在表1433中添加了储存温度信息。
- 修正了少量拼写及格式问题。
- 更新现有错误条目以反映修复状态。
- 更新FREQA.DS0\_RANDOM、FREQA.DS1\_RANDOM、CLK\_SYS\_CTRL.SRC及CLK\_SYS\_CTRL.AUXSRC寄存器的复位值，以反映RP2350 A3的变更。

## 2025年2月20日

- 新增错误修订。

## 2024年12月4日

- 更新寄存器数据。

## 2024年10月16日

- 澄清了M33执行时序中部分 *and*与*and/or*逻辑。

## 2024年10月15日

- 修正了少量拼写及格式问题。
- 新增M33指令时序章节。
- 添加了推荐晶振的链接。
- 记录GPIO总线保持器模式（第9.6.1节），此前仅在SDK文档中描述。

## 2024年9月6日

- 增强版RP2350-E9勘误说明，附加详细信息。
- 改进了调试与追踪组件的描述。
- 修正若干细微拼写错误。

- 实施了若干细微的可读性改进。
- 修正了Minimum Arm **IMAGE\_DEF** 描述中的措辞不准确。

## 2024年8月8日

- 首次发布。



Raspberry Pi is a trademark of Raspberry Pi Ltd

[Raspberry Pi Ltd](#)